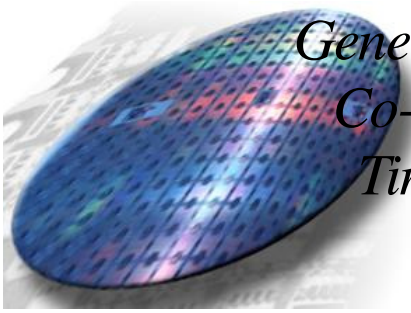
Berner Fachhochschule

Hochschule für Technik und Architektur

University of Applied Sciences Berne

(Biel School of Engineering and Architecture)

The GECKO System



General Purpose Hardware/Software Co-Design Environment for Real-Time Information Processing in System-on-Chip Solutions

Marcel Jacomet, Jörg Breitenstein, Markus Hager, William Chigutsa

1. INTRODUCTION 2

1.1. GENERAL 2
1.2. THE GECKO SYSTEM 2
1.3. SOC 2
1.4. NEW DESIGN METHODOLOGY 3
1.5. THE GECKO DESIGN AND ANALYSIS ENVIRONMENT 4
1.6. GECKO SIGNAL PROCESSING DESIGN AND ANALYSIS CONFIGURATIONS 5

2. THE GECKO SYSTEM 9

2.1. GECKO MAIN BOARD 9
2.2. GECKO EXPANSION BOARDS 11
2.2.1. GECKO I/O Expansion Board 12
2.3. DESIGN & ANALYSIS TOOLS, IP CORES 13

3. GECKO TUTORIALS 14

3.1. TUTORIAL PREPARATION 14
3.1.1. Tutorial Directory Structure 14
*3.1.2. Tutorial **GECKO** Main Board Preparation 15*
3.2. A SIMPLE RTL DESIGN AS SOC 15
3.2.1. Design Description: GeckoTutorial1 15
3.2.2. Design Execution: GeckoTutorial1 16
3.3. A SIMPLE MICROPROCESSOR DESIGN AS SOC 17
3.3.1. Design Description: GeckoTutorial2 17
3.3.2. Design Execution: GeckoTutorial2 18
3.4. A SIMPLE HW/SW CO-DESIGN AS SOC 18
3.4.1. Design Description: GeckoTutorial3 18
3.4.2. Design Execution: GeckoTutorial3 20
3.5. ADVANCED HW/SW CO-DESIGNS AS SOC 20

4. GECKO USER MANUAL 21

4.1. HOST INITIATED BOOT PROCEDURE 21
4.1.1. Overview 21
*4.1.2. Details: Loading the **USB** drivers 21*
4.1.3. Details: Downloading the SoC design 23
4.1.4. Details: SoC system start / restart 25
4.2. PUSH BUTTON INITIATED BOOT PROCEDURE 26
4.2.1. Overview 26
4.2.2. Details: Booting the SoC hardware 26
4.2.3. Details: SoC system restart 27
4.3. JUMPERS, SWITCHES, BUTTONS AND LEDS 28
4.3.1. Overview 28
4.3.2. Details 28
4.4. CONNECTORS 30
4.4.1. Overview 30
4.4.2. Detail: Interface connectors to host computers 30
4.4.3. Detail: Connectors to expansion boards 31
4.5. INTERFACE USB TO SOC 32
4.5.1. Overview 32
4.5.2. Details 33

5. APPENDICES 35

5.1. SCHEMATICS TO THE GECKO MAINBOARD V2.0 35

6. REFERENCES 39

1. Introduction

1.1. General

The GECKO system is a **ge**neral purpose hardware/software **co**-design environment for real-time information processing for system-on-chip (SoC) solutions. The GECKO system supports a new design methodology for system-on-chips, which necessitates co-design of software, fast hardware and dedicated real-time signal processing hardware. Within the GECKO system, the GECKO main board represents an experimental platform, which offers the necessary computing power for speed intensive real-time algorithms as well as the necessary flexibility for control intensive software tasks. The fast prototyping experimental hardware platform represents a key step towards final system-on-chip solutions by verifying the SoC implementation in the real-time environment and by analyzing the processed data.

1.2. The GECKO System

The GECKO system is composed of different hardware boards and software tools. The core element of the GECKO system is the GECKO main board. In addition to the main board, an application specific GECKO expansion board has to be added, if sensors, actuators, power elements or other analog electronic circuitries are part of the target SoC solution. A key element of the GECKO system is the tight link of the GECKO main board to various state-of-the-art design tools. Dedicated drivers, IP (intellectual property) blocks and compilers are responsible to bridge the gap between selected commercial available design and analysis tools and the GECKO main board.

1.3. SoC

Generally speaking, a system can be described by three parts: a user, an algorithm and a plant. The challenge of the designer is to model the interaction between the three parts, to model the plant, and sometimes even to model the user. Once the models are verified, algorithms have to be developed and implemented into an SoC, realizing the interaction as described. A block diagram of a general system is shown in Figure 1. The plant may have different actuators and sensors interacting with some sort of chemical, mechanical or other physical processes. The modelling of the plant may be quite a complex task. The block algorithm in Figure 1 is our electronic system, able to process the sensor and user data as well as able to control the actuators and generate the user information. This generalized system-model will serve us throughout the present manual to describe the different design and analysis steps.

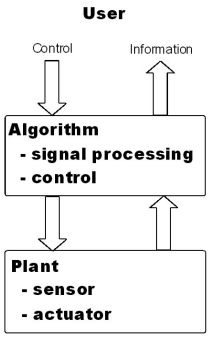


Figure 1: Block diagram of a generalized system. The system is composed of three elements, the user, the algorithm and the plant.

Today's SoC technologies offer sophisticated potentialities. Power elements, sensors, actuators, analog blocks and digital systems including software can be integrated on one single chip. Sometimes, a two-chip solution is preferred due to economical reasons, separating the digital part from the power or analog part. The GECKO system supports the two-chip approach. A large FPGA supports fast prototyping of the digital system part, and the expansion board serves as application specific hardware for the power, sensor or actuator part of the target system (see Figure 2).

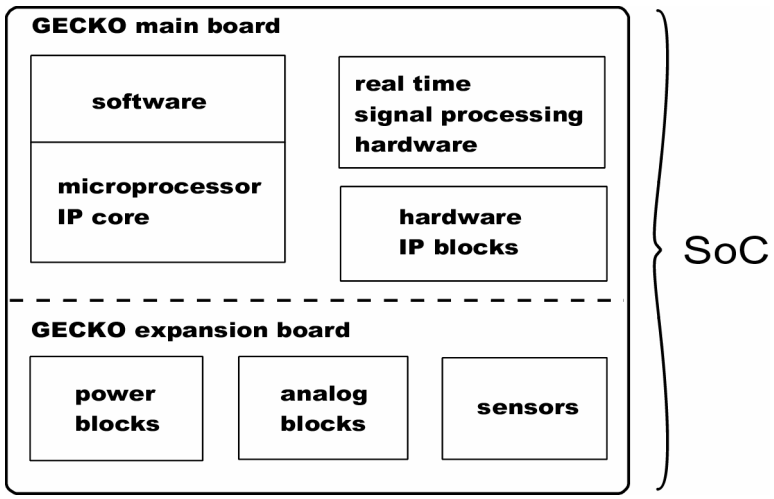


Figure 2: Block diagram of a general SoC. The GECKO system supports the two-chip approach by placing the digital and software subsystem on the GECKO main board and the power, sensor or actuator elements on an application specific GECKO expansion board.

In the following text, the SoC always refers to the hardware/software (HW/SW) system the designer is going to develop and to implement into the GECKO boards.

1.4. New Design Methodology

In data-flow applications, e.g. digital signal processing, the behavior of the system is scheduled at a fixed rate, and the main complexity of the design comes from the mathematical operations on data. As long as precise models of the data to be processed are not available, as is often the case, a straightforward design approach is impossible or at least risky. The flexibility of iteratively improving analog/digital hardware implemented algorithms is thus necessary in order to explore the architecture and to refine its implementation. This involves monitoring capabilities of the SoC

in its real-time environment. Thus an important problem in the classical ASIC hardware/software co-design approach can be identified as the decoupling of the design steps from the prototyping hardware in the design flow, which inhibits the necessary monitoring task in an early design phase. The modern design flow *specify-explore-refine* can easily be adapted to the needs for real-time information processing (see Figure 3). Adding an additional prototyping step and an analysis step opens the potentiality to iteratively improve the SoC design. The *refine-prototype-analysis* design loop matches best the needs of SoC designs with real-time constraints. If precise models of the data to be processed are missing, thorough data analysis and subsequent refinement of the data processing algorithms is a must for designing high quality SoC. To conclude, the *specify-explore-refine* design flow should be extended to a *specify-explore-refine-prototype-analyze* design flow for SoC designs with real-time constraints as it is provided by the GECKO system (see [JGBH01]).

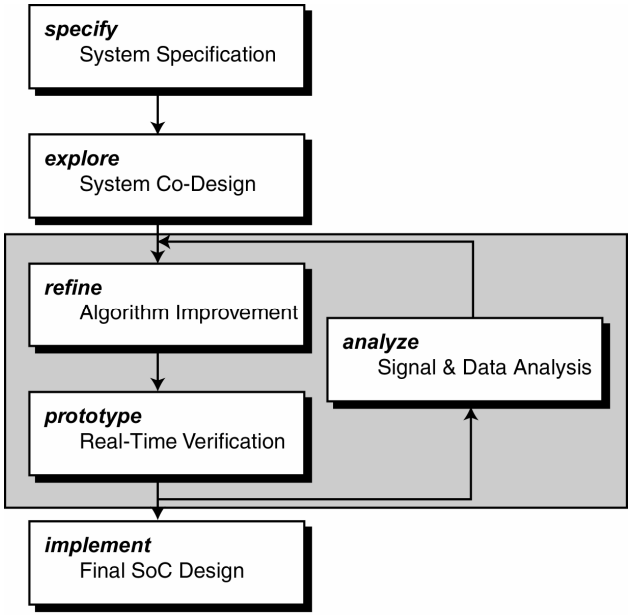


Figure 3. The specify-explore-refine design flow is extended to a specify-explore-refine-prototype-analyze design flow for SoC designs with real-time constraints.

1.5. The GECKO Design and Analysis Environment

The GECKO system uses commercially available, state-of-the-art design and analysis tools for its real-time hardware/software design environment. This approach guaranties the user to be able to use the most sophisticated and widely accepted tools from the designer's community. Figure 4 illustrates the concept of the GECKO real-time information processing and development environment for SoC solutions.

The design entry is done by the three design environments. Software programming is mainly used for control-intensive tasks. In our case we use the C programming language, an IP core out of a set of microprocessor hardware models, and a C compiler. General purpose, speed-intensive tasks are preferably implemented in register-transfer-logic (RTL) hardware using a hardware description language (HDL). Signal processing tasks with real-time constraints can be developed with the Matlab/Simulink environment on a very comfortable high level. The separation of the three design entry points according to their functionality provides the best possible abstraction level for the SoC design. All three sub-designs are converted to VHDL, compiled to a net-list, and placed and routed to the target FPGA on the GECKO main board. The tight link between the HW/SW co-design flow

and the general purpose main board can be seen in Figure 4. Data analysis of a running prototype is a very important and crucial design step in our fast prototyping design methodology. Therefore, an additional data-capture hardware block is available on the GECKO main board. Critical data can now be captured on-line and can be analyzed by Matlab/Simulink off-line in order to iteratively improve the algorithm design by analyzing real world data (see [JGBH01]

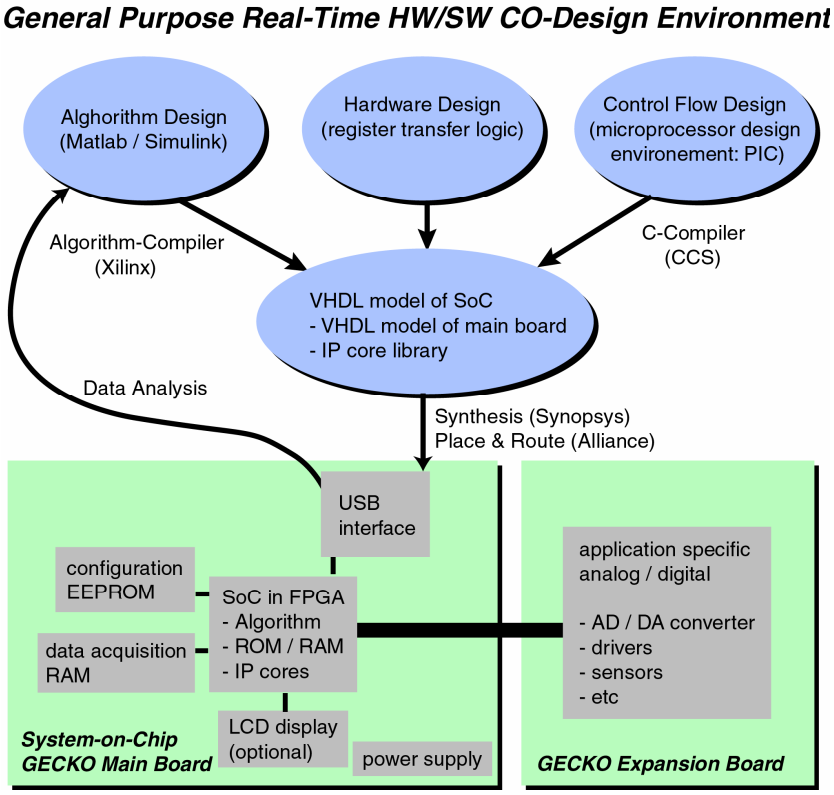


Figure 4: GECKO general purpose real-time HW/SW co-design environment. The SoC design can be developed partially in C for the software tasks, in VHDL for the hardware tasks and in Matlab/Simulink models for the signal processing algorithms. The complete design can subsequently be compiled into an FPGA on the GECKO main board.

1.6. GECKO Signal Processing Design and Analysis Configurations

As shown in the previous sections, the new design methodology asks for a high flexibility in the configuration of the design steps. Fast prototyping and data analysis design steps are necessary in order to iteratively refine and improve the designed algorithm. The decision which part of the system should be analyzed on the highest possible abstraction level and which part needs to be implemented as a fast prototype in the real world strongly depends on the nature of the plant as well as on the maturity of the signal processing algorithm itself.

Case A: Algorithms with plants that can easily be modeled are preferably be analyzed on the highest abstraction level possible as a whole. This highest abstraction level is Matlab/Simulink. This case represents the closed loop simulation of the system on the high abstraction modeling level (see Figure 5).

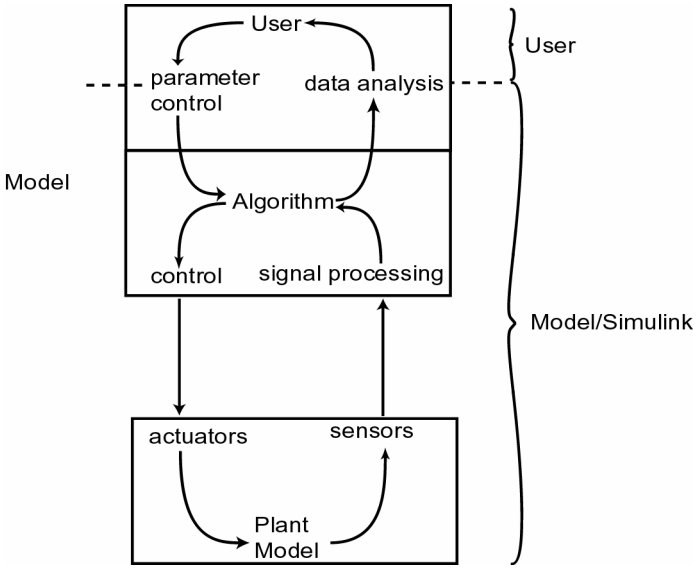


Figure 5: Closed-loop configuration for simulation modeling only.

Case B: Algorithms, with plants which are difficult to model, are preferably first – at least partially - be implemented on a high abstraction level using Matlab/Simulink, directly controlling the real world actuators and sensors. This configuration is only possible if the actuators and sensors in the plant are not dependent on each other and thus the sensor data can be captured off-line. This configuration represents an open loop with real world data provided for the algorithm modeling level. The GECKO system fully supports this case (see Figure 6).

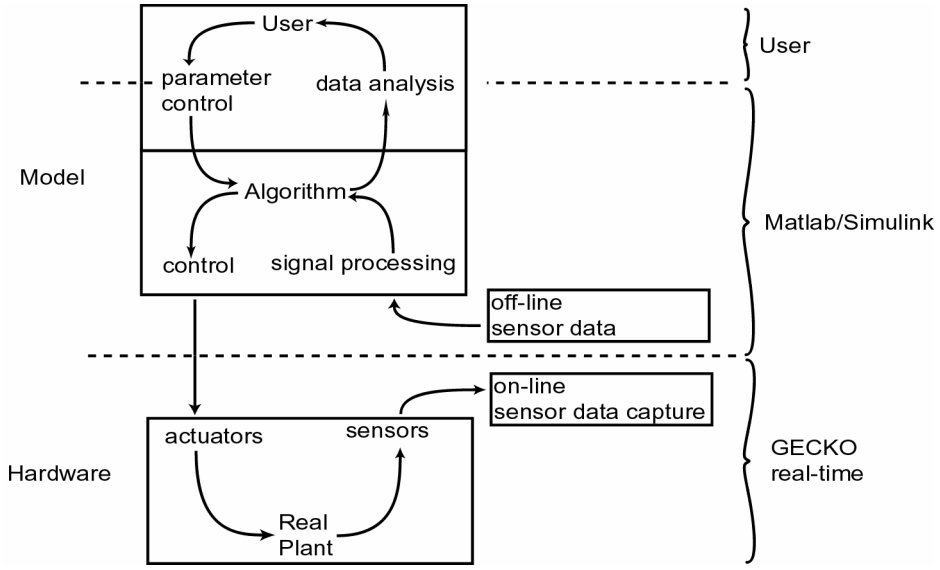


Figure 6: Open loop configuration for algorithm implementation with real sensor data.

Case C: As described in case B, algorithms, with plants which are difficult to model, are preferably first (partially) be implemented on a high abstraction level using Matlab/Simulink, directly controlling the real world actuators and sensors by a hardware-in-the-simulation loop. In case the real-time constraints are not too strict (sampling rates are not too high), this configuration can be realized using Matlab/Simulink for the algorithmic part and the GECKO boards for controlling the actuators and capturing the sensor data. The GECKO system fully supports this case as long as the real-time constraints are not exceeding the Matlab/Simulink speed limitations. This case is also called hardware-in-the-simulation loop and is subject in our current research work (see Figure 7).

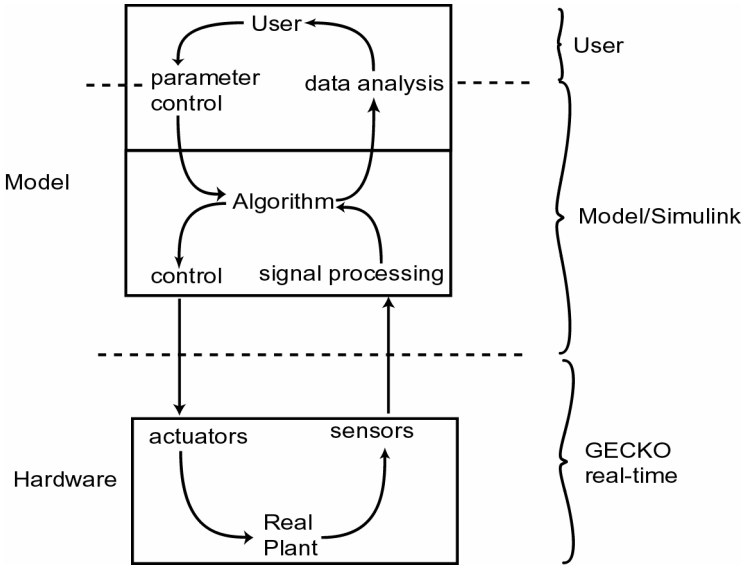


Figure 7: Closed loop configuration for algorithm implementation with real sensor data (hardware-in-the-simulation-loop).

Case D: Once first algorithm solutions are developed on the Matlab/Simulink environment, the algorithms can be implemented and executed as fast prototypes on the GECKO main board. With the on-line data capture feature of the GECKO main board, critical data can be stored locally and be analyzed off-line on the Matlab/Simulink environment for data analysis purposes and further algorithm improvement. This configuration represents a fast prototype closed-loop system implementation. Again the GECKO system fully supports this case (see Figure 8, see also [JGHC02]).

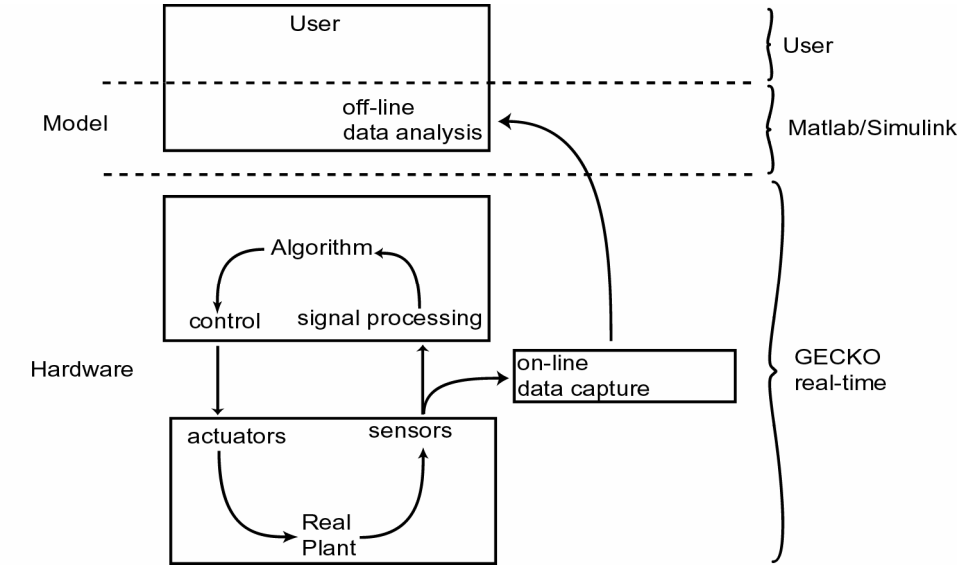


Figure 8: Closed loop configuration for fast prototyping implementation.

Case E: Once first algorithm solutions are developed on the Matlab/Simulink environment, the algorithms can be implemented and executed as fast hardware-in-the-loop prototypes on the GECKO main board. With the on-line data capture feature of the GECKO main board, critical data can be analyzed on-line on the Matlab/Simulink environment for data analysis purposes and further algorithm improvement. In addition, parameters can be changed on-line in the Matlab/Simulink environment and immediately be updated on the running hardware. This configuration represents a fast prototype closed loop system implementation. This configuration is also called the hardware-in-the-loop system implementation. The described hardware-in-the-loop system implementation is subject of our current research work and is currently running with limitations only.

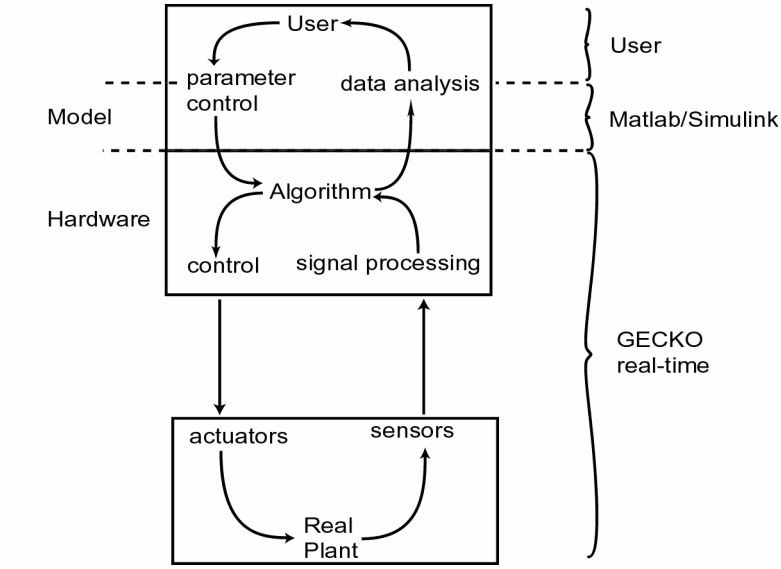


Figure 9: Hardware-in-the-loop configuration for fast prototyping implementation.

2. The GECKO System

2.1. GECKO Main Board

The GECKO boards consist of two type of boards: the GECKO main board and the application specific GECKO expansion boards. The GECKO main board is based on a large RAM-based FPGA (Spartan II family from Xilinx Inc, XC2S100 or XC2S200 chips). The FPGA is supposed to hold the prototype versions of the complete digital part of the system-on-chip solution. Important is the additional on-board data capture RAM of the GECKO main board, which is used to store the sampled data from the expansion board for the signal analysis phase. An USB interface chip (EZ-USB from Cypress) with an on-chip 8051 microprocessor interfaces our SoC FPGA of the main board with a host computer. Large sets of the sampled data can be uploaded through the USB interface into the Matlab/Simulink development tools. The offer of such an off-line processing of real data is very important in order to analyze the source data and to devise refinements for the real-time information processing algorithms. In iterative steps the refined algorithms can again be downloaded to the SoC FPGA. Such a closed-loop refinement and analysis of real-time information processing systems is crucial to improve the understanding of the nature of the data to be processed as well as the necessary algorithms.

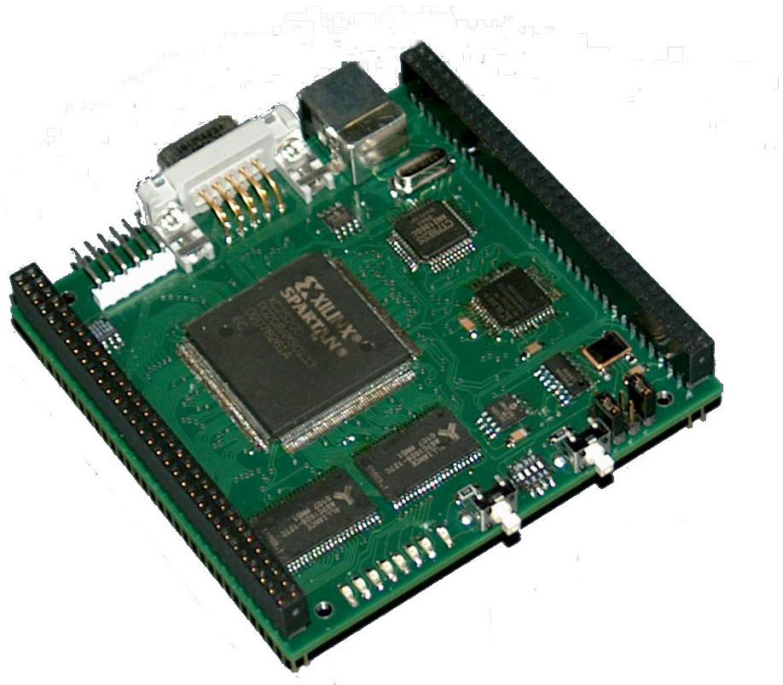


Figure 10: Photo of the GECKO main board with a size of 8cm x 8cm.

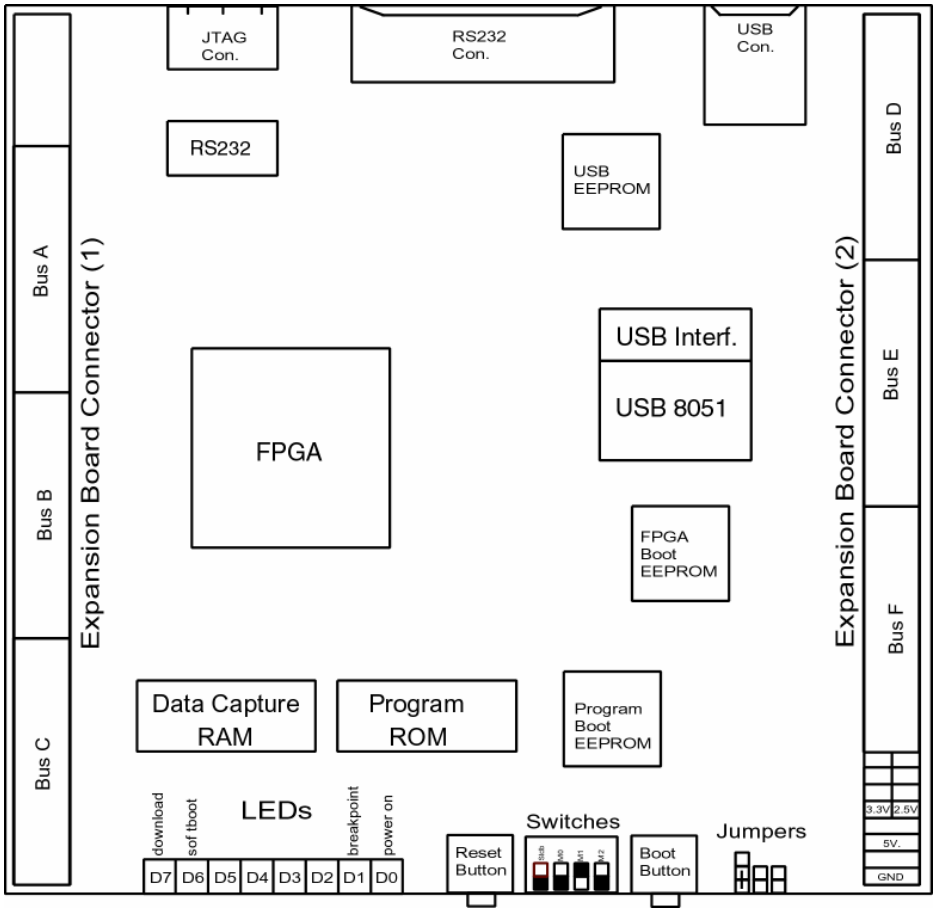


Figure 11: Block diagram of the GECKO main board.

The following list describes all the components of the GECKO main board as illustrated in Figure 11.

- **FPGA**: The Xilinx FPGA XC2S200 is a 200k logic gate FPGA used to house the SoC target system (some systems house the smaller XC2S100 FPGA).
- **FPGA Boot EEPROM**: The SoC hardware configuration file for the target FPGA can either be directly booted (hardware boot) from this parallel EEPROM or be downloaded by the host computer on the fly (see jumper J2 configuration)
- **USB Interface**: The USB interface to the host computer is used to download the FPGA configuration files, to download the compiled microprocessor program as well as to download and upload data from the GECKO main board to the host computer.
- **USB EEPROM**: This serial I2C EEPROM (16k x 8 bit) houses the identification code and parameters for the USB driver.
- **USB 8051** processor: The firmware of the USB driver on the GECKO main board is running on its on-chip 8051 processor.
- **RS232**: The RS232 interface can be used by the SoC application for communication purposes.
- **Program ROM**: The Program ROM is implemented as a RAM. This external 64k x 16bit RAM can be used for the microprocessors application software of the SoC if the FPGA internal RAM is too small to store the microprocessor program code.

- **Program Boot EEPROM**: This 32k x 8 bit serial I2C EEPROM is used to store the software program code for the embedded microprocessor. During the software boot procedure it is copied to the Program ROM for the microprocessor.
- **Data Capture RAM**: The 64k x 16 bit RAM can be used for the data capture feature of the GECKO board or for in the SoC application itself.
- **Clock Generator**: An on board quartz clock generator can be used to clock SoC system. The USB block has its own and independent clock.
- **Power Supply**: The USB interface delivers a 5V power supply, which is used by the GECKO main board. The necessary 3.3V and 2.5V supply voltages are generated on-board. The current consumption from the USB power supply is limited as defined in the USB specifications.
- **Expansion Board Connectors**: Two 64 bit connectors are used to send and receive signals from and to the GECKO expansion boards. Six 16 bit buses *BusA* to *BusF* can be used with some limitations. Some control signals are also routed to the Expansion Board Connectors like the on board SoC clock signal *sysclk*, the expansion board clocks *extclk1* and *extclk2* and the on-board *reset*. The on board I2C bus signals *scl* and *sda* are also available on the Expansion Board Connectors and can be used after the software boot process is finished. System ground and all power supply voltages are routed to the Expansion Board Connectors as well.
- **LEDS**: 4 LEDs are used for system information (power on, software boot process, download, breakpoint), the other 4 LEDs as well as 2 of the system information LEDs (software boot process, breakpoint) can be used freely by the SoC.
- **Push Buttons**: A boot push button is used to initiate the hardware boot process. The second push button activates a *reset* signal, which is freely usable by the SoC for resetting purposes and/or software boot.
- **Jumpers and Switches**: Several jumpers and SMD switches are used to define different main board configurations, like the use of the external clocks for SoC, FPGA configuration by host or on-board EEPROM, 5V power supply from main board routed to expansion board.

2.1.1. Files

Pin constraints file:
groups\microlab\technologie\MicroLab-lib\IPcores\GeckoMisc\pin**GECKOpinning.ctl**

Schematics of the board can be found in the appendices at the end of this document.

2.2. GECKO Expansion Boards

Application specific analog signal processing, analog interfaces, sensors, or power drivers have to be placed on expansion boards which can be stacked on our general purpose main board. Numerous buses can be used to exchange data between the two boards. Important is the additional on-board RAM of the main board, which is used to store the sampled data from the expansion board for the signal analysis phase.

2.2.1. GECKO I/O Expansion Board

The idea of this board combined with the GECKO main board is to have a simple development environment to do some exercises.

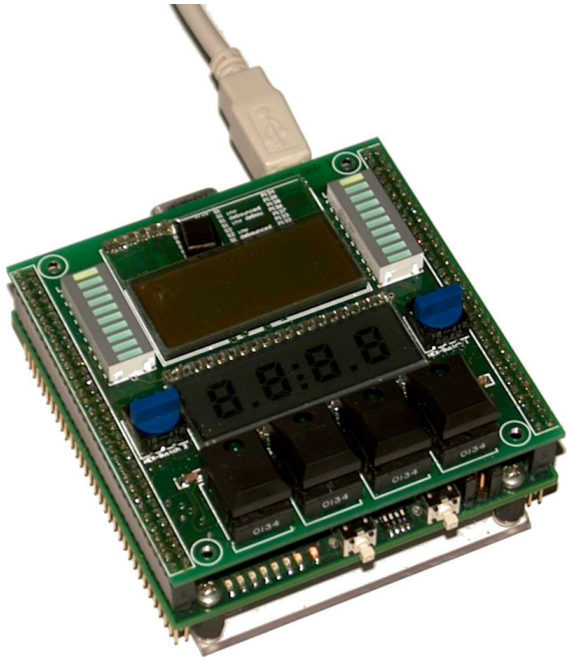


Figure 12: Photo of the I/O Expansion Board

Technical Data:

- **4 switches: N**ormal, inverted and debounced output.
- **2 Hex-Switches:** which create directly the binary code.
- **2 LED arrays:** Each with ten LEDs.
- **1 LCD 7-segment display:** 4 digits (HINT: The LCD-7-seg frequency must be 32 Hz).
- **1 I^2C character display**

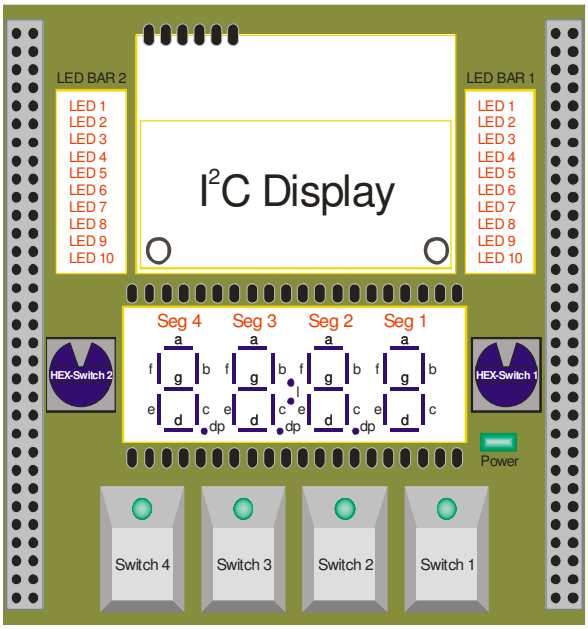


Figure 13: Block diagram of the I/O EXPANSION Board

\groups\microlab\projekte\intern\GECKO\expansion_board\
IOExpansion Board**DokuExpansion_Board_Connectors.xls**

2.3. Design & Analysis Tools, IP Cores

Some of the GECKO design and analysis tools can easily be replaced by the users own preference, others have a very tight link to the GECKO system and cannot be replaced without major changes. The following tools, drivers and hardware boards are currently used within the GECKO design and analysis environment:

- **Software Development**: The C-Programming environment from the company [CCS] Inc. is used. The tools include a C editor, C libraries and a compiler for the PIC series microprocessors from Microchip Inc. ([Mic]).
- **Microprocessor IP Cores**: Several PIC microprocessors from Microchip Inc. have been modeled as VHDL IP cores by MicroLab-I3S. Currently the following PICs are available:
  - PIC16C5X series for limited program memory size (on-chip RAM)
  - PIC16C7X series for large program memory size (external RAM)
- **Hardware Development**: To develop digital hardware the VHDL language is used together with the Synopsys Inc. tools for entry (SGE tool), simulation (VSS tool) and compilation (design analyzer).
- **Signal Processing**: The Matlab/Simulink tools from Mathworks Inc. ([Mat]) are a de-facto standard for design and analysis of signal processing algorithms and are thus used for real-time constraint driven hardware implementations of the algorithms. Xilinx Inc offers a special Toolbox for the Matlab/Simulink environment with signal processing blocks which can be compiled to VHDL code using the System Generator tool from Xilinx Inc .([Xil]). The synthesis of the generated VHDL code is currently executed by the compiler from Synplicity Inc. ([Syn]) instead of the one by Synopsys (recommended by Xilinx).
- **Back-End**: The place and route tasks of the synthesized VHDL netlist is done by the Xilinx Alliance tools.
- **Download**: The Xilinx FPGA configuration file of the digital circuit as well as the compiled C software program can be downloaded into the GECKO main board using the GECKO User Interface provided by MicroLab-I3S.
- **Data Upload, Off-Line Analysis**: With the GECKO data capture feature on the main board any signal or sensor data can be captured on-line and be stored into a date capture RAM on the GECKO main board. The data can be uploaded to the Matlab/Simulink environment for an off-line analysis. As the on-line data capture is application specific, an adaptation of the GECKO driver provided by MicroLab-I3S is necessary.
- **Hardware-In-The-Loop**: Currently research is done in order to include the GECKO main board into the Real-Time-Workshop from Mathworks. This would allow us to run signal processing models partly on Matlab/Simulink and partly on the GECKO main board in real time. Parameter change during runtime and on-line data analysis would open new debugging and monitoring possibilities.

3. GECKO Tutorials

3.1. Tutorial Preparation

3.1.1. Tutorial Directory Structure

All source files of the following tutorials are available. In addition all necessary executable files for downloading to the GECKO main board are already generated and available for the user. Tool dependent design or configuration files are only available for the tools used at MicroLab-I3S which are Synopsys sge schematic editor and fc2 FPGA synthesis tools, Xilinx alliance place&route and SystemGenerator tools, Mathworks Matlab/Simulink tools, CCS C compiler tool. It is up to the user to manage his design tools and being able to edit the necessary tool configuration files. Information like VHDL signal to FPGA pin mapping is in addition prepared. For each of the tutorial designs the following directory structure is available:

Gecko/	root of GECKO relevant design and library files
tutorials/	root directory for all tutorials
GeckoTutorial1/	root directory for tutorial 1
vhdl/	VHDL source files
pin/	pin location file (for Xilinx tools)
Ccode/	C source code of embedded microprocessor program
hex/	C source compiled to hex file with machine code
bit/	Xilinx FPGA configuration bit files
mat/	Matlab/Simulink source files
tools/	ASIC design tool dependent files
xilinx/	Xilinx place & route tool files
synopsys/	Synopsys sge schematic design entry and fc2 synthesis
GeckoTutorial2/	root directory of tutorial 2
… …	
IPcores/	root directory for MicroLab-I3S IP cores
PIC16C7Xseries/	root directory of Microchip PIC16C7X microprocessors
src87/	VHDL files of microprocessor IP cores
sym/	Synopsys SGE design entry schematic files of IP cores
lib/	complied IP core library for Synopsys VSS simuator
PIC16C5Xseries/	root directory of Microchip PIC16C5X microprocessors
src87/	VHDL files of microprocessor IP cores
sym/	Synopsys SGE design entry schematic files of IP cores
lib/	complied IP core library for Synopsys VSS simuator
GeckoBoot/	root directory of GECKO boot and misc IP blocks
src87/	VHDL files of IP cores
sym/	Synopsys SGE design entry schematic files of IP cores
lib/	complied IP core library for Synopsys VSS simuator
PICsoftLib/	C code libraries for GECKO specific functions

Not all designs need all 3 types of input source files for VHDL, C code or Matlab/Simulink and thus some of the corresponding directories *vhdl*, *Ccode* or *mat* may be empty.

If you want to execute the tutorials on the GECKO board you just need to copy the compiled files in the tutorial directories called bit and hex to your host account. If you want to put the source design files through your own CAD tools and then run the tutorials on the GECKO main board, you need to copy the above design and library files and add the necessary tool configuration files prior to synthesize the designs.

3.1.2. Tutorial GECKO Main Board Preparation

In all tutorials the jumpers and switches have to verified and set as described in the Figure 14. In addition be sure, that the on-board quartz oscillator is running somewhere between 5 MHz and 15 MHz for good results. It is not relevant which type of Xilinx FPGA is mounted on the board, but you should know its type in order to select or generate the correct configuration file. There are currently two versions available with the following FPGA mounted, Spartan2 XC2S100 or XC2S200 both in a QFP208 socket.

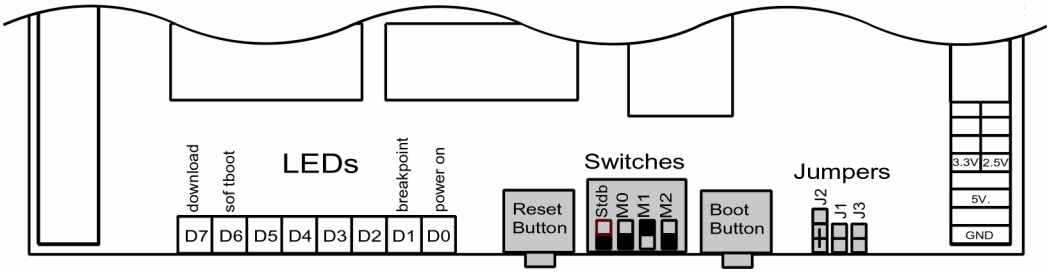


Figure 14: Jumpers and switches setting for all tutorials (to describe an example, the Standby switch is set in position up).

3.2. A Simple RTL Design as SoC

3.2.1. Design Description: GeckoTutorial1

A first very simple design of a "SoC" for the FPGA, called GeckoTutorial1, running on the GECKO main board is described. The "SoC" is a simple clock divider, used as a 4 bit binary counter connected to 4 LEDs of the GECKO main board. The "SoC" design only consists of a digital hardware part, not using any microprocessor (see block diagram in Figure 15).

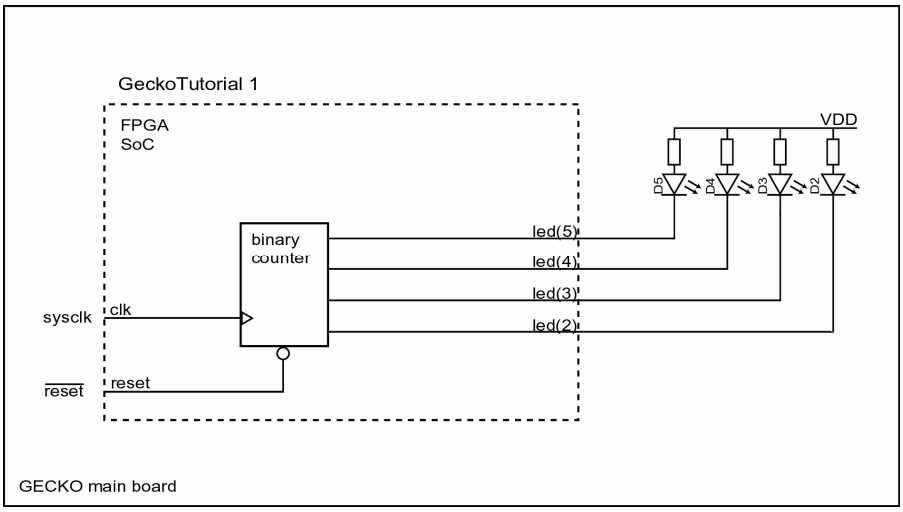


Figure 15: Block diagram of the tutorial GeckoTutorial1 design.

The design is done by means of an RTL (register transfer logic) description based VHDL code, which has been designed manually, simulated, compiled, placed & routed to the target FPGA technology (XC2S100 or XC2S200 Xilinx FPGA) on the GECKO main board. As a classical chip design flow is used, it is left to the designer to execute these steps on his own design tools. In the following Figure 16, the necessary source and result files for this first design are illustrated.

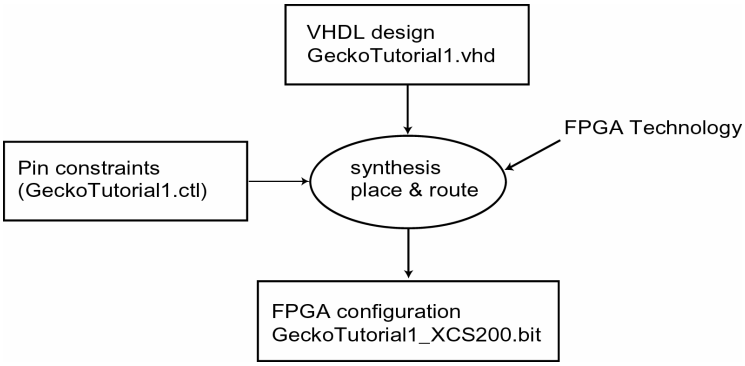


Figure 16: Design source and result files of the GeckoTutorial1 example.

3.2.2. Design Execution: GeckoTutorial1

In order to execute the tutorial on the GECKO main board, the user has only to download the FPGA configuration file *GeckoTutorial1_XC2S200.bit* to the GECKO main board (housing a XC2S200 QFP208 pin chip as a target FPGA). The following steps have to be done:

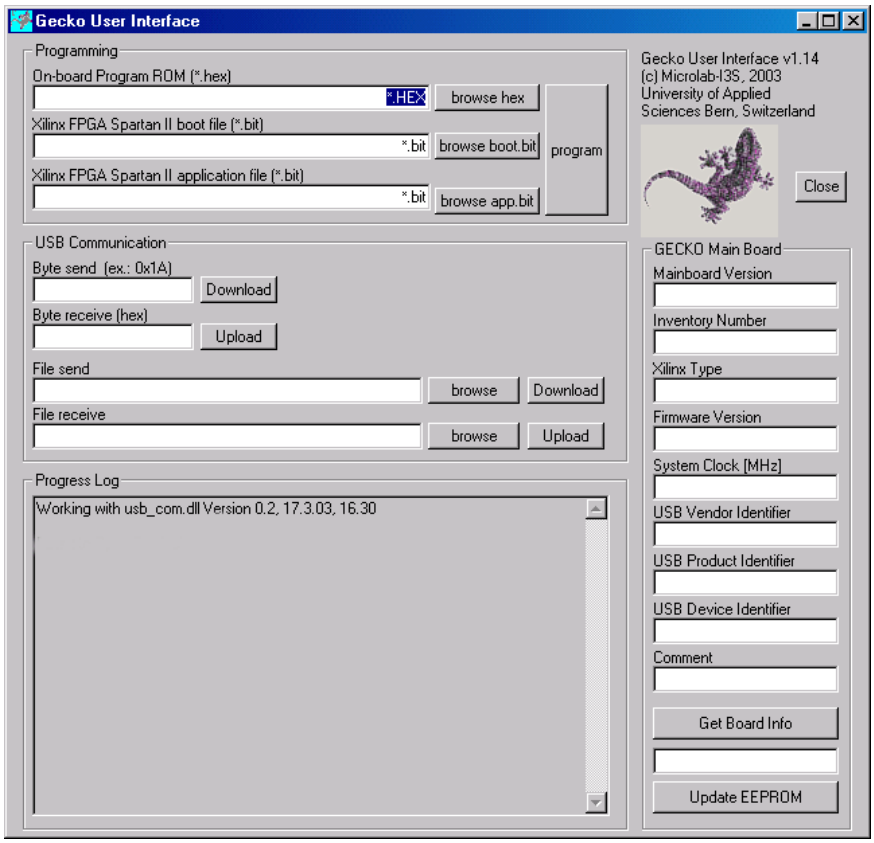


Figure 17: GUI of the GECKO User Interface running on the host computer.

Step 1: Connect the GECKO main board with an USB port of the host computer. The power on LED D0 is on.

Step 2: Run the GECKO User Interface on the host computer.

Step 3: Load your "SoC" design configuration bit file *GeckoTutorial1_XC2S200.bit* into the GECKO User Interface by using the **Browser** button in the Programming section of the drivers GUI (see Figure 17).

Step 4: Download the file by using the **Program** button in the Programming section of the drivers GUI. The LED D7 will be on for a fraction of a second, signaling the hardware downloading. Quit the GECKO User Interface.

Step 5: Press the reset Push Button SW2 on the GECKO main board to start the application correctly. Quite often the application may start without pressing the reset Push Button, as the FPGA internal registers are always reset to logic '0' values after loading the SoC design.

As the binary counter of the application is connected to the LEDs D5 downto D2, you will see the binary output value counting up for a new value exactly every second for system clock frequency of 13.56 MHz.

3.3. A Simple Microprocessor Design as SoC

3.3.1. Design Description: GeckoTutorial2

A second very simple design of a "SoC" for the FPGA, called GeckoTutorial2, running on the GECKO main board is described. The "SoC" is a microprocessor IP core of the Microchip PIC16C5X processor series. Four bits of the PIC16C57 microprocessor internal port A (bits A0 to A3) are configured as output signals and connected to 4 LEDs of the GECKO main board. The LEDs are up/down counting the first seconds, and then flashing very fast for the following seconds. The "SoC" design only consists of the PIC16C57 microprocessor hardware IP core and an embedded software program running on the PIC placed in an FPGA internal Program ROM and not on the FPGA external Program ROM chip of the GECKO main board (see block diagram in Figure 18).

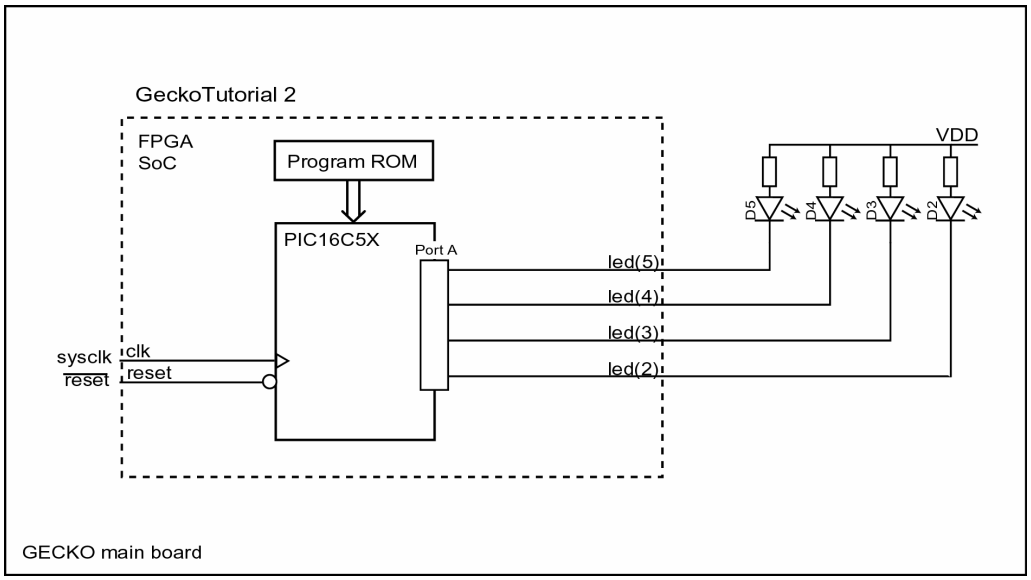


Figure 18: Block diagram of the tutorial GeckoTutorial2 design.

The design is done by means of a simple C software program, which was designed manually, simulated, and compiled to the target PIC16C57 microprocessor, resulting in a hex file for the compiled microprocessor machine code. In the following Figure 19, the necessary source and result files for this second design are illustrated.

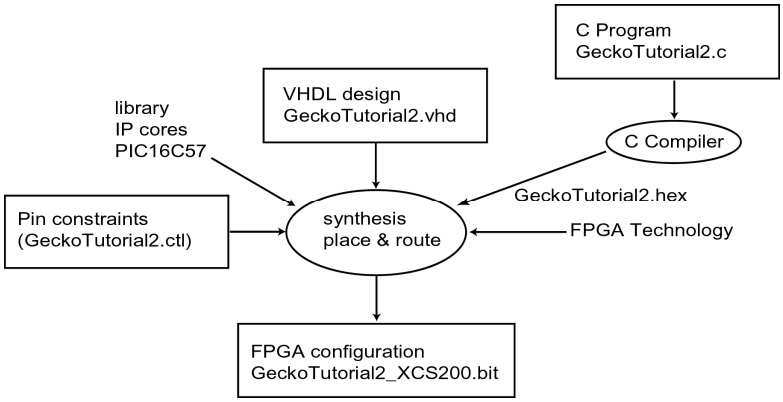


Figure 19: Design source and result files of the GeckoTutorial2 example.

3.3.2. Design Execution: GeckoTutorial2

In order to execute the tutorial on the GECKO main board, again the user has only to download the FPGA configuration file *GeckoTutorial2_XC2S200.bit* to the GECKO main board (housing a XC2S200 QFP208 pin chip as a target FPGA). In order to run the tutorial design on the GECKO main board, the same steps as in GeckoTutorial1 have to be executed.

Note that the configuration bit file contains the PIC microprocessor as well as the contents of its internal program memory, which in our case is the compiled machine code (hex file) of the C program. This means that if a new C code program is compiled, a complete design flow, starting from VHDL synthesis down to place & route to the SoC target FPGA has to be performed.

3.4. A Simple HW/SW Co-Design as SoC

3.4.1. Design Description: GeckoTutorial3

A third simple design of a "SoC" for the FPGA, called GeckoTutorial3, running on the GECKO main board is described. This time the "SoC" is composed of a hardware block and a microprocessor IP core modeling the Microchip PIC16C7X processor series, using an external Program ROM. In this tutorial a pseudorandom number generator is implemented as SoC. In this third application the user can communicate with the SoC through the GECKO User Interface running on the host computer. The user can download commands to the SoC and upload results (SoC commands: 00 for stopping the pseudorandom generator, 01 for running the pseudorandom generator, 02 for single pseudorandom value reading, 03 for continuously pseudorandom values reading). The SoC commands can be downloaded to the SoC by the byte send command of the GECKO User Interface. The results are available at the GECKO User Interface in the byte receive window for the SoC single reading mode or with the file open/close command for the SoC continuously reading mode. The pseudorandom generator is implemented as a dedicated hardware block, the control logic and communication to the USB interface are implemented in software, running on the embedded microprocessor.

Three bits of the hardware random pattern generator are connected to 3 LEDs of the GECKO main board, signaling to the user its operation mode (LED D5 for pseudorandom generator stopped, LED D4 for pseudorandom pattern generator running, LED D3 for reading single or continuously random value, on for 1 second for each reading). The PIC port E is connected to the USBdata bus for data exchange with the USB 8051, the port D (D0 to D3) is used for the necessary handshake signals. The PIC port B is used to interface with the hardware block generating pseudorandom values, port A (bits A0 to A2) is used to control the communicate with the pseudorandom pattern generator

hardware block. The embedded software program is stored on the FPGA external Program ROM chip of the GECKO main board (see block diagram in Figure 20).

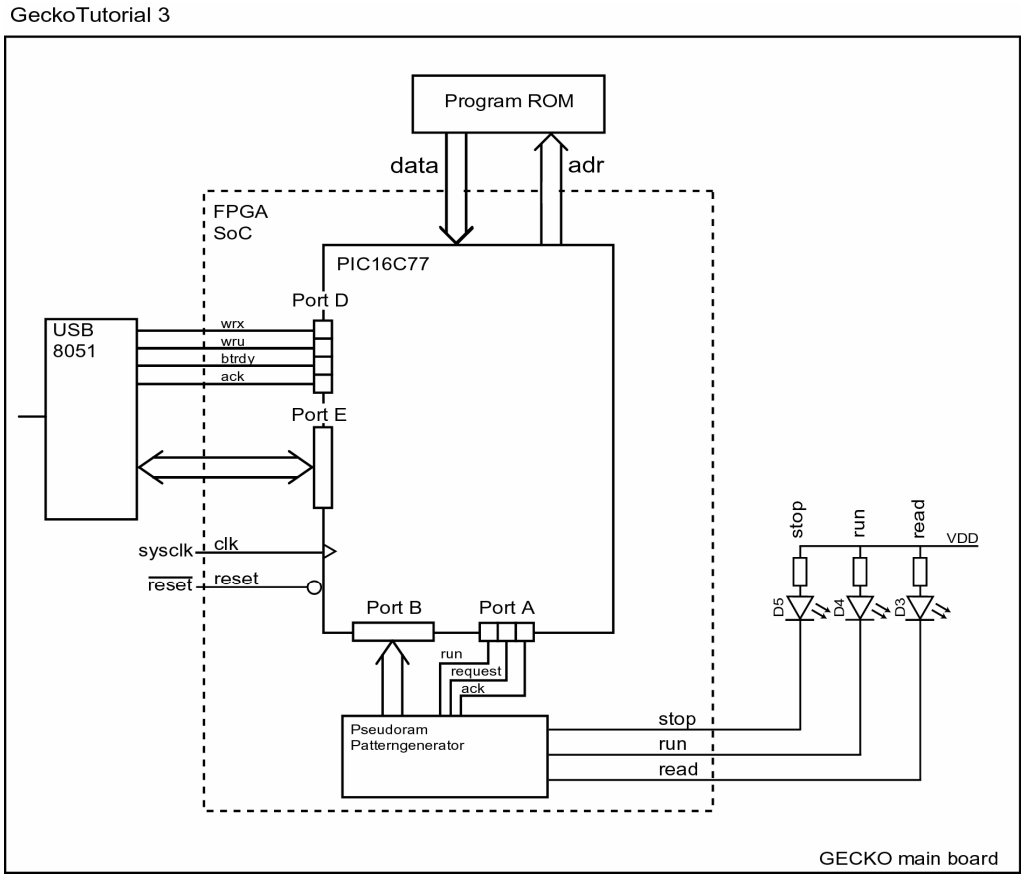


Figure 20: Block diagram of the tutorial GeckoTutorial3 design.

The design is done by means of a hardware block, manually designed with VHDL code and a simple C software program, which was developed, simulated, and compiled to the target PIC16C77 microprocessor, resulting in a hex file for the compiled microprocessor machine code. In the following Figure 21, the necessary source and result files for this second design are illustrated.

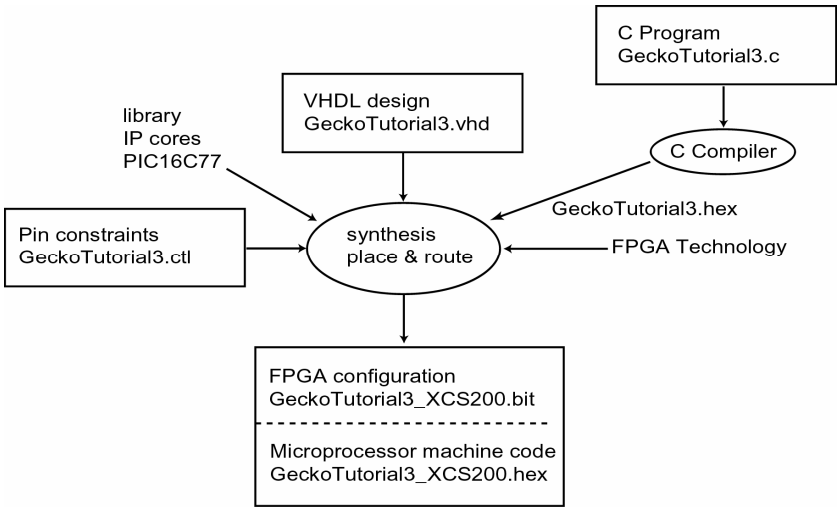


Figure 21: Design source and result files of the GeckoTutorial3 example.

3.4.2. Design Execution: GeckoTutorial3

In order to execute the tutorial on the GECKO main board, the user first has to download the compiled hex file of the embedded PIC processor to the on-board Program RAM *GeckoTutorial3.hex* followed by the FPGA configuration file for the GECKO main board *GeckoTutorial3_XC2S200.bit* (housing a XC2S200 QFP208 pin chip as a target FPGA). In order to run the tutorial design on the GECKO main board, the following steps may be executed.

Step 1: Write the command value 01 (pseudorandom generator running) into the byte send window of the GECK User Interface. Send the command value to the SoC. The LED D4 will activate indicating that the pseudorandom generator is running.

Step 2: Write the command value 02 (pseudorandom generator single reading) into the byte send window of the GECK User Interface. Send the command value to the SoC. A pseudorandom value will be uploaded into the byte receive window of the GECKO User Interface. Repeat step 2 to get new pseudorandom values. The LED D3 will be activated at each reading for a one second, indicating that the SoC is reading a single value.

Step 3: For continuously reading pseudorandom values, a file has to be opened with the GECKO User Interface. Then the command 03 (continuously pseudorandom values reading) has to be downloaded to the SoC. The SoC will generate now pseudorandom values and upload them into the open file at the host. The LED D3 will be on indicating that the SoC is continuously reading.

Step 4: To finish continuously reading pseudorandom values, send the command 00 (stop pseudorandom generator) to the SoC and then close the file with the GECKO User Interface. The LED D5 will be on, indicating a stop of the pseudorandom generator.

Note that the configuration bit file does not contain the PIC microprocessor anymore. This means that if a new C code program is compiled, the hardware synthesis has not to be re-invoked and only the new hex file needs to be downloaded, followed the FPGA configuration bit file.

3.5. Advanced HW/SW Co-Designs as SoC

Additional tutorials will be developed in the near future which will use a new application specific tutorial GECKO expansion board. The expansion board will hold elements like buttons, LCD interface, audio in and audio out interface as well as H bridges for driving dc motors. Signal processing applications can then be designed on the Matlab/Simulink level and downloaded to the GECKO main board holding the tutorial GECKO expansion board.

4. GECKO User Manual

4.1. Host Initiated Boot Procedure

4.1.1. Overview

The GECKO main board is used as an experimental and development board. This asks for a highly flexible hardware and software system. In order to free the user from the complicated configuration steps, several automatic boot procedures are implemented. The first one, the host initiated boot procedure, is started automatically upon connecting the USB interface of the GECKO main board to the host computer. This host initiated boot procedure is mainly used during the development phase of the SoC design. The following 3 phases to download a design from the host to the GECKO main board are implemented:

- Loading the USB drivers
- Downloading the SoC hardware and the embedded microprocessor software program.
- Resetting the SoC hardware and booting the embedded microprocessor (software booting) for system start.

The following two conditions have to be satisfied for a correct host initiated boot procedure:

- Jumper J2 has to be in position **host** , switch SW3 has to be in position (Stdby=**off**, M2=**off**, M1=**on**, M0=**off**)
- The necessary SoC configuration and program files have to be generated.

Depending on the SoC design, an interaction of the SoC design with the boot process might be necessary in order to find an optimal solution for the SoC. Knowing the detailed host initiated boot procedure, the designer can even adapt the default boot procedure and thus develop an application specific host initiated boot procedure.

4.1.2. Details: Loading the **USB** drivers

The previously described phase to load the USB drivers of the host computer as well as of the GECKO main board and thus setting up the connection between host computer and GECKO main board, is actually composed of the following sequence of 10 steps (see Figure 22):

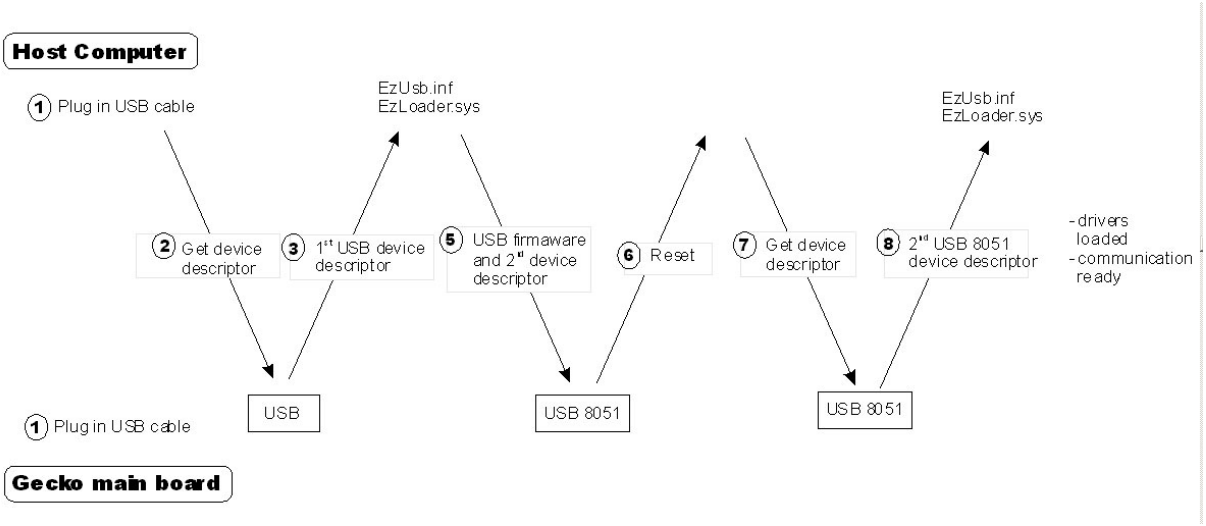


Figure 22: Procedure to load the USB drivers on the host computer and the GECKO main board.

Step 1: Manual connection of the host computer to the GECKO main board by plugging in the USB interface cable. The LED D0 will be activated to indicate the power-on status of the GECKO main board.

Step 2: The general purpose USB driver of the host operating system (Windows 95/98/ME/2000/NT/XP) will detect the presence of a new USB device and ask for its identity (get descriptor) which is stored in the so called USB device descriptor.

Step 3: The USB device on the GECKO main board sends its hard coded device descriptor (1st device descriptor), which contains information like vendor identification (Cypress) and product identification (EZ-USB) to the host computer.

Step 4: The host computer gets the USB device descriptor from the GECKO main board and searches the host computer operating systems INF files for a match. A positive identification of the USB device descriptor will be possible with the EZUSB.INF driver configuration file from the host computer. Further activities will now be initiated according to the EZUSB.INF file.

Step 5: The host computer will now load the boot driver EZLOADER.SYS according to the USB device descriptor and the EZUSB.INF file information. The boot driver downloads the application firmware for the GECKO USB 8051 device.

Step 6: After storing the firmware in its internal nonvolatile memory, the GECKO USB 8051 device executes an internal reset, which has the same effect as plugging in the USB cable at step 2.

Step 7: The general purpose USB driver of the host operating will again detect the presence of a new USB device and ask for its USB device descriptor.

Step 8: The firmware of the USB device on the GECKO main board sends its new device descriptor (2nd device descriptor), which contains information like vendor identification (MicroLab-I3S) and product identification (GECKO main board) to the host computer.

Step 9: The host computer gets the 2nd USB device descriptor from the GECKO main board and searches the host computer operating systems INF files for a match. A positive identification of the USB device descriptor will again be possible with the EZUSB.INF driver file from the host computer. Further activities will thus be initiated according to the EZUSB.INF file.

Step 10: The host computer will now load the default application driver EZUSB.SYS according to the 2nd USB device descriptor and the EZUSB.INF file information.

Host computer and GECKO main board are now able to communicate with each other, based on the communication functions made available from the host computer driver EZUSB.SYS and the USB 8051 firmware on the GECKO main board.

4.1.3. Details: Downloading the SoC design

There are principally two approaches to setup the SoC design in the GECKO main board. A first approach is to download all design relevant information from a host computer. The only precondition for this approach is the presence and setup of a USB connection as well as the correct setting of jumper J2 in position **host**. In the second approach the GECKO main board can setup itself in a stand-alone mode, without using any USB connections. This second approach will be described in a next sub-chapter.

The following steps describe in detail the downloading of the SoC design from the host computer to the GECKO main board. The steps are illustrated using a data flow model of the GECKO main board. It is the designer's responsibility to identify possible conflicts of the download procedure and his running SoC design.

Step 1: The Jumper J1 has to be in position **off**. This de-connects the system clock from the FPGA cclk pin, which is used for configuration file loading.

Step 2: The Jumper J2 has to be in position **host**. This de-connects the on-board FPGA Boot EEPROM from the FPGA and thus permits configuration file loading for the SoC from the host (hardware boot).

Step 3: Switch SW3 has to be set to the following positions: Stdby=**off**, M2=**off**, M1=**on**, M0=**off**. This selects the FPGA mode slave parallel for loading the configuration file from the host.

Step 4: Run the application program GECKO User Interface on the host computer: GECKOdriver.EXE

Step 5: A GUI will appear on the host computer, where the FPGA configuration file and the compiled software program can be selected for downloading to the GECKO main board.

Step 6: If the SoC design is a mixed hardware/software co-design, then the internal microprocessor of the SoC (IP microprocessor core) needs an embedded software program to be executed and stored in some memory on the GECKO board. This embedded C software program for the microprocessor IP core has to be compiled prior to downloading and is thus available in form of a machine code (hex file) for the program memory on the GECKO main board. Select your embedded SoC software program file and download it. A sample embedded SoC software program is available as a compiled hex file: GeckoExample3.hex

Step 7: The GECKO download application will now download the hex file through the USB interface to the GECKO main board. The USB 8051 firmware receives the hex file of the embedded software program and stores it into the Program Boot EEPROM (32k x 8 bit) via the serial I2C interface. As the embedded software program is stored in a nonvolatile EEPROM, it has only to be downloaded when new versions of the embedded software have been developed.

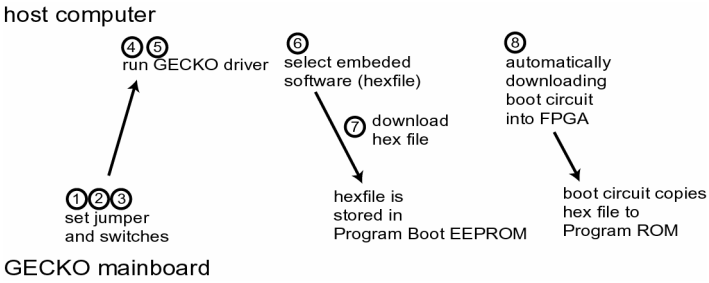


Figure 23: Downloading of the embedded software program (hex file) to the Program Boot EEPROM.

Step 8: After downloading the embedded software for the SoC, the software program needs to be copied from the Program Boot EEPROM to the Program ROM (which is actually a RAM) via the I2C interface. A special IP software boot circuit has been developed for the FPGA holding the necessary hardware to copy the program code from the Program Boot EEPROM to the Program ROM. After the hex file downloading, the GECKO User Interface will automatically download this IP software boot circuit (bit file) into the FPGA (LED D7 is on during downloading, hardware boot) and start its execution. LED D6 is on during the hex file copying procedure to its final destination (software boot), which is the Program ROM. Previously downloaded SoC designs will now not be available anymore and have again to be downloaded to the FPGA.

Step 9: The FPGA configuration file (bit file) is holding the digital hardware part of the SoC design. If the SoC design is a mixed hardware/software co-design, then the microprocessor IP core had also to be included in the SoC design Select your SoC design file with the GECKO User Interface. Sample tutorial designs are available as a compiled configuration bit files: GeckoExample1.bit to GeckoExample3.bit.

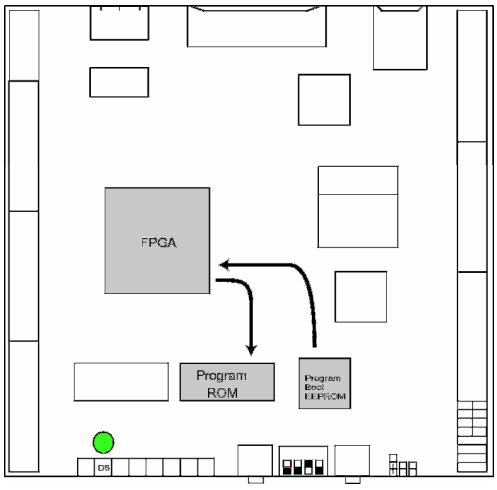


Figure 24: Software boot procedure: Transferring the embedded software program from the Program Boot EEPROM to the software program memory (Program ROM).

Step 10: Download your SoC design (bit file) with the GECKO User Interface through the USB interface to the GECKO main board. The USB 8051 firmware receives the FPGA configuration bit file and sends it to the FPGA. The USB 8051 configures the FPGA by using the fast parallel configuration mode of

the Xilinx FPGA. During downloading LED D7 will be activated for a very short time period.

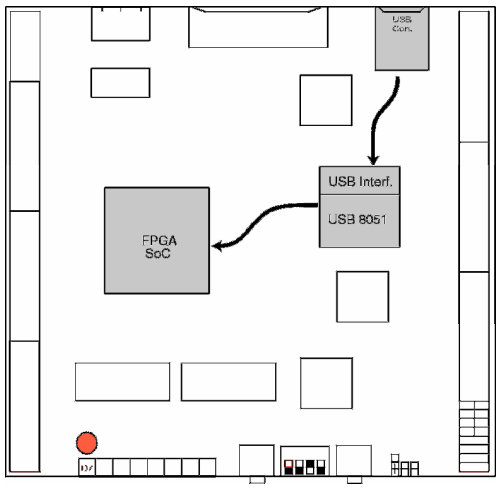


Figure 25: Hardware boot procedure: Downloading the configuration file (bit file) to the FPGA.

4.1.4. Details: SoC system start / restart

After completion of the USB driver loading and the SoC downloading phases, finally the SoC can start its execution.

Step 11: Pressing the Push Button SW2 will activate the reset of the SoC, the embedded microprocessor of the SoC design will be released from its reset state and the processor starts its normal execution. The SoC is now running. The PIC IP cores will start their normal execution without an external activation of the reset line as their program counter is initiated the starting address of the software program (address = 0).

As soon as the host initiated boot procedure is terminated, the I2C bus as well as the data bus connecting the USB 8051 to the FPGA are released and available for the SoC.

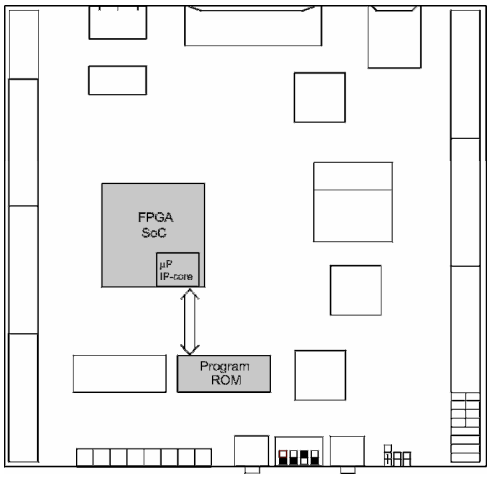


Figure 26: After completing of all downloading/booting procedures, the SoC is running. The illustration shows a SoC with an embedded IP core using an external program memory.

4.2. Push Button Initiated Boot Procedure

4.2.1. Overview

In addition to the host initiated boot procedure, a so called push button initiated boot procedure can be used. The push button initiated boot procedure is mainly used after completion of the design, as no program or configuration files need to be downloaded from the host to the GECKO main board. In case the SoC application does not need any USB interface, the push button initiated boot procedure can be implemented for the SoC design. During the push button initiated boot procedure the GECKO main board will boot itself automatically, using the stored configuration and program files from the two on-board boot EEPROMs.

The following phases are implemented:

- Booting the SoC hardware at power-on-reset.
- Loading the embedded microprocessor software program (for PIC16C7X series) and resetting the SoC hardware for system start.

The following conditions have to be satisfied for a correct host initiated boot procedure:

- Jumper J2 has to be in position **onboard** , switch SW3 has to have the position (Stdby=**off**, M2=**off**, M1=**on**, M0=**off**).
- One of the IP software boot blocks has to be included in the SoC design. This is in contrast to the host initiated boot procedure, where no IP software boot block has to be included in the SoC design.
- The two boot files are already stored in the on-board boot EEPROMs.
- If the USB interface is not connected, an external power supply of 5V is necessary. In this case the Jumper J3 USBPOWER has to be set.

4.2.2. Details: Booting the SoC hardware

In order to be able to use the push button initiated boot procedure, the boot files have already to be stored in the two on-board EEPROM (see loading the USB drivers).

Step 1: The Jumper J1 has to be in position **off**. This de-connects the system clock from the FPGA cclk pin, which is used for configuration file loading.

Step 2: The Jumper J2 has to be in position **onboard**. This connects the on-board FPGA EEPROM to the FPGA for loading the SoC configuration file.

Step 3: Switch SW3 has to be set to the following positions: Stdby=**off**, M2=**off**, M1=**on**, M0=**off**. This selects the FPGA mode master serial for loading the configuration file from the FPGA EEPROM.

Step 4: The push button initiated boot procedure starts automatically upon power-on-reset. Press the Push Button SW1 for a short time period. This will re-start the configuration file loading from the FPGA Boot EEPROM into the FPGA. During the loading procedure, LED D7 will be activated (fraction of a second), indicating the end of load when LED D7 is de-activated again. The SoC is now loaded into the FPGA.

Step 5: If a microprocessor IP core with an external Program ROM is used then the software boot block included in the SoC will copy the program hex file from the serial Program Boot EEPROM to the Program ROM. Upon completion a reset to the SoC design is issued. It is the designer's responsibility to add the

software block into his SoC design and to connect the SoC design with the boot reset line. LED D6 is on during the software copying process.

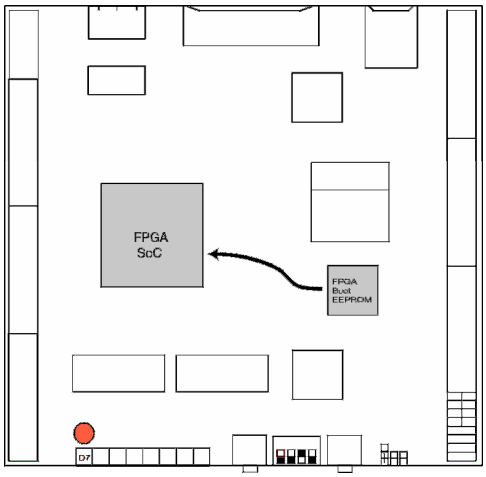


Figure 27: Hardware boot procedure: Configuration file (bit file) is loaded from the FPGA Boot EEPROM into the FPGA.

As soon as the push button initiated boot procedure is terminated the data bus connecting the USB 8051 to the FPGA are released and available for the SoC. LEDs D1and D3 are now de-activated.

4.2.3. Details: SoC system restart

After completion of the SoC hardware and software boot phase, the SoC can again be restarted.

Step 1: Depending of the SoC design, the Program ROM of the embedded microprocessor is located internally of the FPGA or on the Program ROM of the GECKO main board. Upon pressing the Push Button SW2 (SoC reset), the IP software boot block in the SoC will read the Program Boot EEPROM through its serial I2C bus and store the embedded software program into the Program ROM. The sample design GeckoTutorial3 uses the on-board Program ROM for the embedded software program. During the software transfer procedure the LED D6 will be activated.

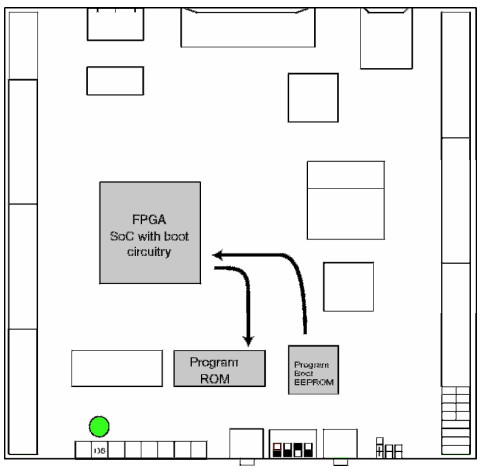


Figure 28: Embedded software program boot procedure: Transferring the embedded software program from the Program Boot EEPROM to the Program ROM.

Step 2: As soon as the embedded software program boot procedure is finished, the embedded microprocessor of the SoC design will be released from its reset state and the processor starts its normal execution. The SoC is now running.

A re-start of the SoC can always be provoked by initiating the embedded software program boot procedure through the Push Button SW2.

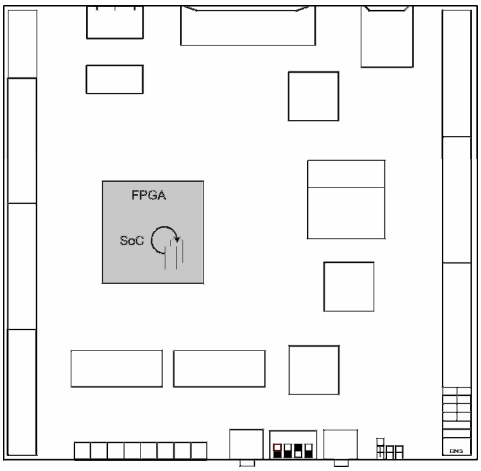


Figure 29: After completing of all downloading/booting procedures, the SoC is running.

As soon as the push button initiated boot procedure is terminated, the I2C bus as well as the data bus connecting the USB 8051 to the FPGA are released and available for the SoC. The SoC is now running and LED D7 as well as D6 are de-activated (see Figure 29).

4.3. Jumpers, Switches, Buttons and LEDs

4.3.1. Overview

In order to keep the GECKO main board flexible, a set of jumpers and switches have to be set correctly. Push buttons are used to initiate activities, LEDs indicate their correct termination:

- Jumpers and switches are used for download/boot configurations.
- Push Buttons are used to initiate boot processes.
- LEDs are used to show download and boot activities.

4.3.2. Details

In a tabular representation all downloading and boot configuration relevant elements are explained in detail. Jumpers and switches are used to configure the GECKO main board downloading and boot procedures:

Name	*Position*	*Function*
Jumper J1	**off**	De-connects sysclk from cclk and thus allows the FPGA to be loaded.
Jumper J2	**onboard** (1-2)	FPGA configuration will be loaded from the GECKO on-board FPGA Boot EEPROM
	host (2-3)	FPGA configuration will be downloaded from the host computer
Jumper J3	**on**	GECKO board will be powered by USB.
Jumper J3	**off**	GECKO board will be powered by expansion boards. USB power supply is not used.
Switch SW3	M2=**on**, M1=**off**, M0=**off**	FPGA configuration will be downloaded from the

		host computer (slave parallel FPGA mode)
	M2=**off**, M1=**on**, M0=**on**	FPGA configuration will be loaded from the GECKO on-board FPGA Boot EEPROM (master serial FPGA mode)
Switch SW3	standby=**off**	normal FPGA mode
	standby=**on**	sets FPGA permanently into standby mode

The Push Buttons are used to initiate the on-board boot procedure.

Name	*Position*	*Function*
Push Button SW1	**press**	Deletes the SoC design in the FPGA. If Jumper J2 is set to position **onboard**, then a new SoC deign (configuration bit file) will be loaded into the FPGA.
Push Button SW2	**press**	Loads embedded microprocessor software from Program Boot EEPROM into Program ROM

The LEDs are used to show power on, ongoing download and boot activities.

Name	*Position*	*Activity Time*	*Function*
LED D0	**red**	immediately	Power on
LED D1	**red**		Reserved for future use (breakpoint). D2 can currently be used by the user (logic '0' for LED on).
LED D2-D5	**green**		Can be used by the SoC (logic '0' for LED on).
LED D6	**green**	2 sec for 8KB	Copying embedded software from Program Boot EEPROM to Program ROM (software boot). D6 can be used by the user if no hardware boot procedure is used within the SoC (logic '0' for LED on).
LED D7	**red**	less than 1 sec	FPGA configuration in progress (hardware boot)

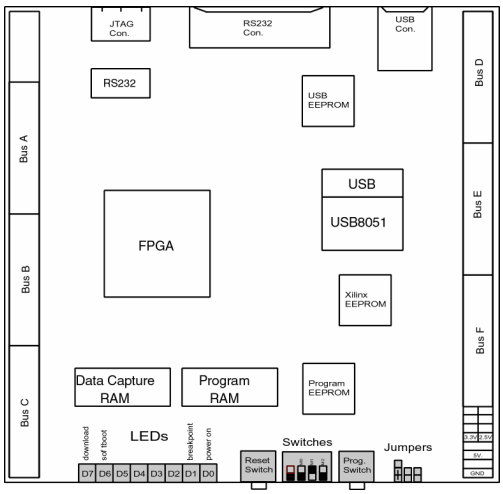


Figure 30: Jumpers, switches, push buttons and LED of the GECKO main board.

4.4. Connectors

4.4.1. Overview

The GECKO main board is used as the digital part of a SoC system. Thus numerous connections need to be foreseen to the GECKO expansion boards, which host the analog part or power part of the overall SoC. Two types of connectors are present on the GECKO main board:

- Connectors for standard interfaces to host computer: USB, RS232, Boundary Scan (JTAG).
- Connectors for system internal buses of the SoC to the expansion boards.

4.4.2. Detail: Interface connectors to host computers

The three interfaces to the host computers serve different purposes and can be used as follows:

- **USB 1.0**: (female B type connector): The interface can be used to download SoC configuration and program files and to initiating the boot process. Once the SoC system is configured, the USB interface can be used by the running SoC system to communicate with the host computer in order to exchange data and commands (see details in the following chapters).
- **RS232**: (9-pin female connector type): Once the SoC system is configured, the RS232 interface can be used by the running SoC system to communicate with the host computer in order to exchange data and commands (see details in the following chapters)
- **Boundary Scan / JTAG** (6 pin Xilinx connector): This interface can be used by a host computer to download the SoC configuration file to the FPGA. Configuration file downloading through the boundary scan interface is an alternative to use the USB interface in case the host does not support the USB interface.

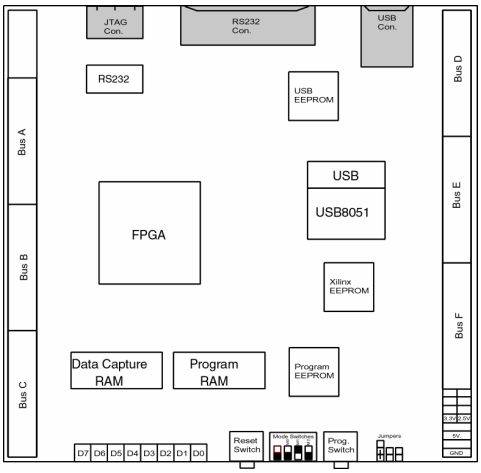


Figure 31: Interface connectors to host computers.

4.4.3. Detail: Connectors to expansion boards

Two 64 pin connectors con1 and con2 on each side of the GECKO main board are used for signal and power supply connections to the GECKO expansion boards. In the following table all pins are described in detail.

Signal Name	Connector	Restriction	Purpose
BusA	con1		connected to I/O of FPGA
BusB	con1	Also used as address bus of Data Capture RAM. BusB is not usable if Data Capture RAM is used.	connected to I/O of FPGA
BusC	con1	Also used as data bus of Data Capture RAM. BusC is not usable if Data Capture RAM is used.	connected to I/O of FPGA
BusD	con2		connected to I/O of FPGA
BusE	con2		connected to I/O of FPGA
BusF	con2	Also used as data bus of FPGA external Program ROM. BusF is not usable if embedded microprocessor uses external Program ROM.	connected to I/O of FPGA
scl, sda	con2	During download and soft boot process the I2C bus is used. Freely available thereafter.	I2C bus connected to USB 8051, to FPGA and serial Boot EEPROMs
sysclk	con2	Clock source is the GECKO main board clock generator (standby turns clock off)	Connected to FPGA GCLK0 clk input. Default SoC system clock from GECKO main board. Can be used to synchronize expansion board with main board.
extclk1, extclk2	con2	Clock sources are the GECKO expansion board clock generators (standby turns clock off at FPGA input)	Connected to FPGA GCLK2, GCLK3 clk inputs, respectively.
reset	con2		Active low reset. Connected to I/O of FPGA and reset Push Button SW2. Used for software boot and SoC reset.
GND	con2		System ground
+5V	con2	Limited current available for expansion board if powered by USB	Power supply from or to main board (see position of J3)
+3.3V	con2	Limited current available for expansion board if powered by USB	Power supply generated at main board
+2.5V	con2	Limited current available for expansion board if powered by USB	Power supply generated at main board

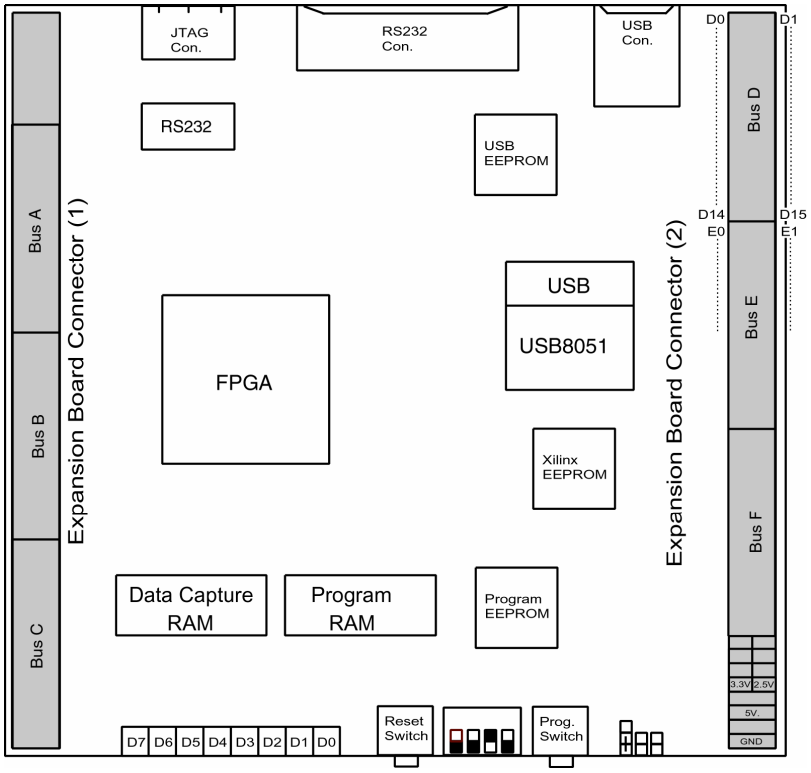


Figure 32: Expansion bus connectors.

4.5. Interface USB to SoC

4.5.1. Overview

The USB interface can be used to download SoC configuration files and initiating the boot process. Once the SoC system is configured, the USB interface can be used by the running SoC system to communicate with the host computer in order to exchange data and commands. The hardware protocol for this interface between USB 8051 and the SoC housed in the FPGA is defined by the firmware in the USB 8051. With the current firmware the communication is realized as follows.2scirocco &

The data to be transferred between port D of the USB 8051 and at I/O on the SoC in the FPGA (see Annex A for detailed pin numbers). Four handshake signals are used for the communication. All handshake signals are active low:

- **wrx** (write Xilinx FPGA): This is a unidirectional signal from the SoC in the FPGA, indicating that the SoC wants to write data to the USB.
- **wru** (write USB 8051): This is a unidirectional signal from the USB 8051, indicating that the USB wants to write data to the SoC.
- **btrdy** (byte ready): This is a bi-directional signal, which is driven by the writing device, indicating that the data is now ready for reading.
- **ack** (acknowledge): This is bi-directional signal, which is driven by the receiving device, indicating that the data has been read.

4.5.2. Details

The hardware communication between USB 8051 and the SoC housed in the FPGA is based on the four handshake signals wrx, wru, btrdy and ack described above. The following illustration (Figure 33) describes in detail the data transfer from the SoC in the FPGA to the USB 8051:

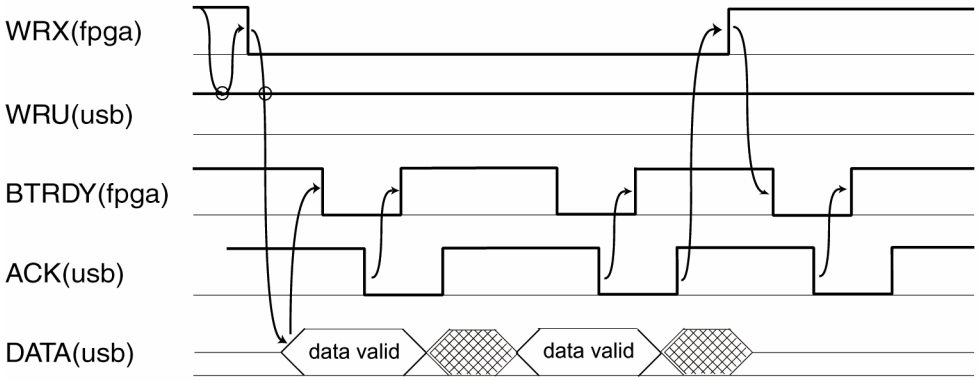


Figure 33: Communication protocoll for data transfer from SoC to USB 8051.

Step 1: Before the SoC device can activate its wrx signal, he has to check if not the USB 8051 has already activated its wru signal before. The SoC activates its wrx in the SoC and drives the btrdy signal. The USB 8051 drives the ack signal.

Step 2: The SoC again has to check if he was the only one requesting to write data. If not both of the devices have to deactivate their requests, release their btrdy and ack signals. They may again try to write.

Step 3: Now the SoC can place its data on the USBbus and thereafter signalling valid data by activating the btrdy signal.

Step 4: The USB 8051 can now read the data from the USBbus and signals its correct operation by activating the ack signal.

Step 5: The SoC deactivates the btrdy signal and thereupon releases the USBbus.

Step 6: The USB 8051 de-activates the ack signal.

Step 7: If the SoC wants to write additional data to the USB 8051, then it repeats steps 3 to 6.

Step 8: Else the SoC terminates the data transfer by de-activating the wrx and releasing the btrdy signal.

Note that the signals btrdy and ack are bi-directional tri-state signals and as the signals wru and wrx they have a pull-up resistor internal to the FPGA.

The data transfer in the other direction, from the USB 8051 to the SoC is identical as described before, except that the roles of the SoC and the USB 8051 are exchanged and thus the signals wrx and wru (see Figure 34).

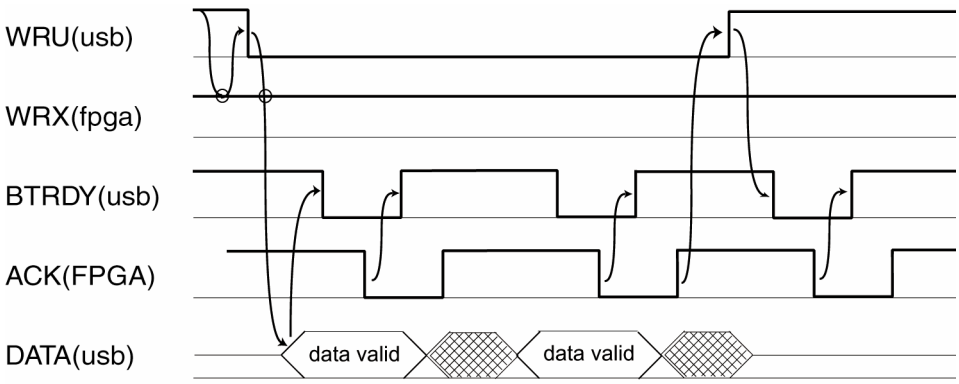
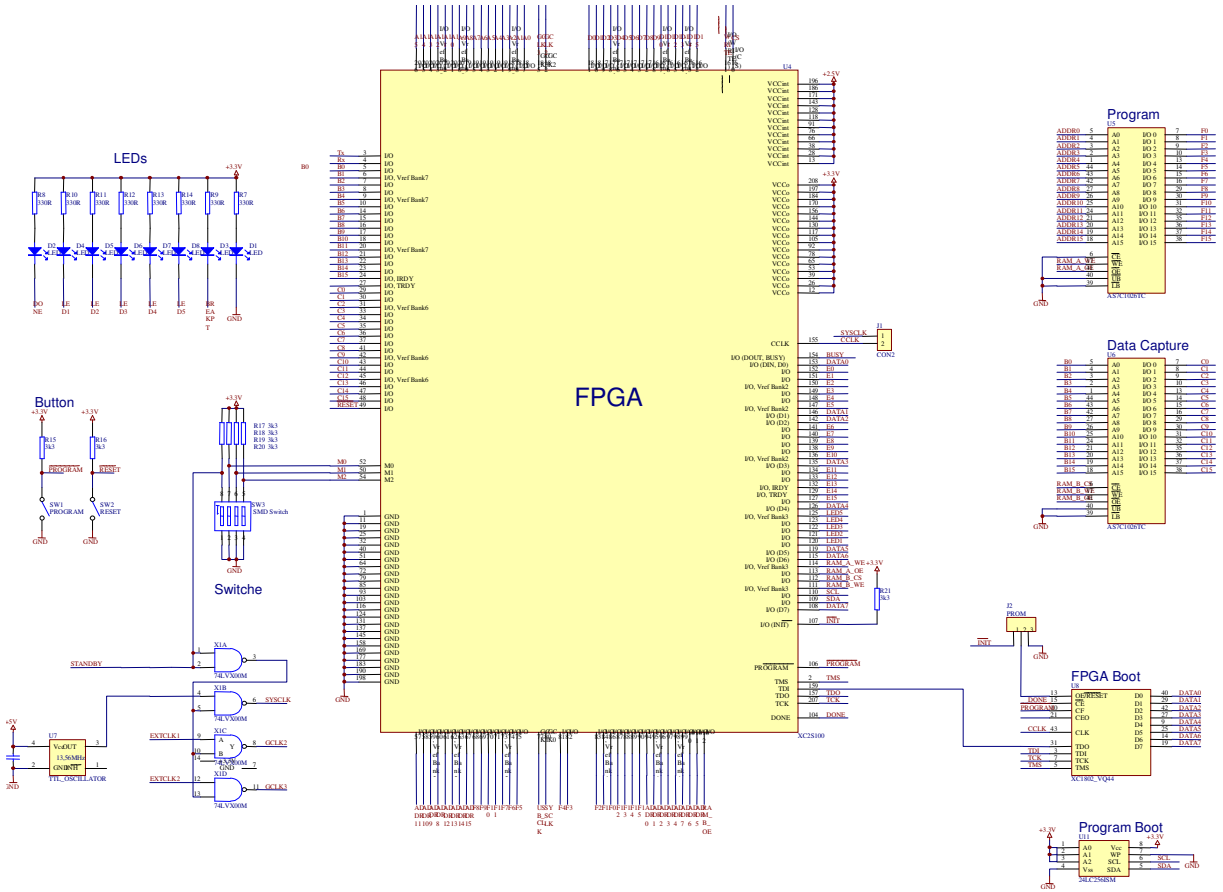
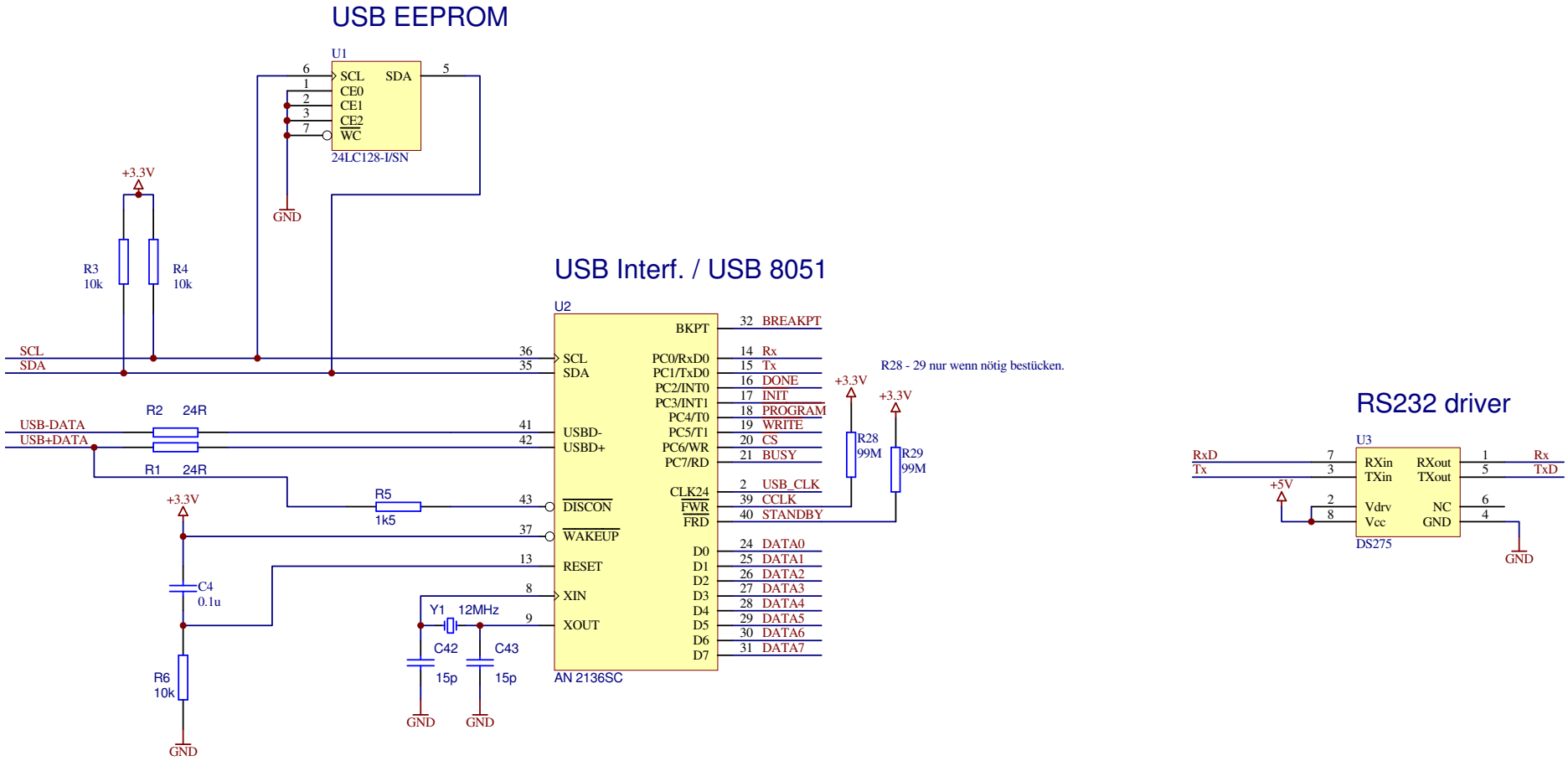


Figure 34: Communication protocol for data transfer from USB 8051 to SoC.

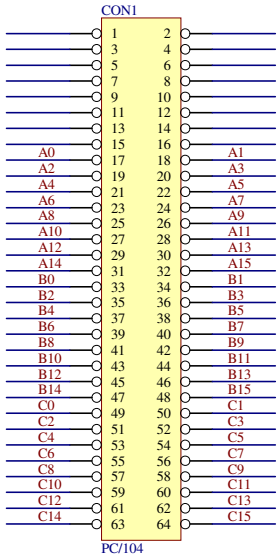
5. Appendices

5.1. Schematics to the Gecko mainboard V2.0

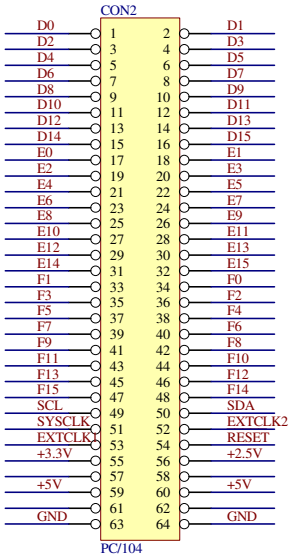



USB EEPROM
U1
SCL
SDA
CE0
CE1
CE2
WC
24LC128-I/SN
+3.3V
GND
R3
10k
R4
10k
USB Interf. / USB 8051
U2
BKPT
SCL
SDA
USBD-
USBD+
DISCON
WAKEUP
RESET
XIN
XOUT
PC0/RxD0
PC1/TxD0
PC2/INT0
PC3/INT1
PC4/T0
PC5/T1
PC6/WR
PC7/RD
CLK24
FWR
FRD
D0
D1
D2
D3
D4
D5
D6
D7
AN 2136SC
BREAKPT
Rx
Tx
DONE
INIT
PROGRAM
WRITE
CS
BUSY
USB_CLK
CCLK
STANDBY
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
R28 - 29 nur wenn nötig bestücken.
R28
99M
R29
99M
SCL
SDA
USB-DATA
USB+DATA
R2 24R
R1 24R
R5
1k5
C4
0.1u
R6
10k
Y1 12MHz
C42
15p
C43
15p
RS232 driver
U3
RXin
TXin
Vdrv
Vcc
RXout
TXout
NC
GND
DS275
RxD
Tx
Rx
TxD
+5V

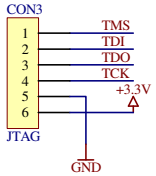
Expansion Board Connector (1)



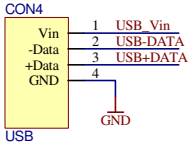
Expansion Board Connector (2)



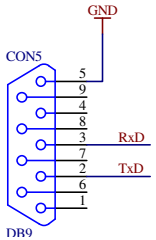
JTAG Connector

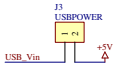


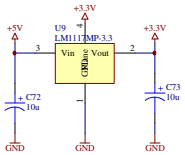
USB Connector

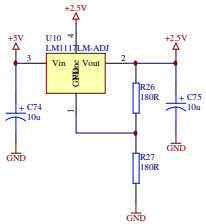


RS232 Connecto

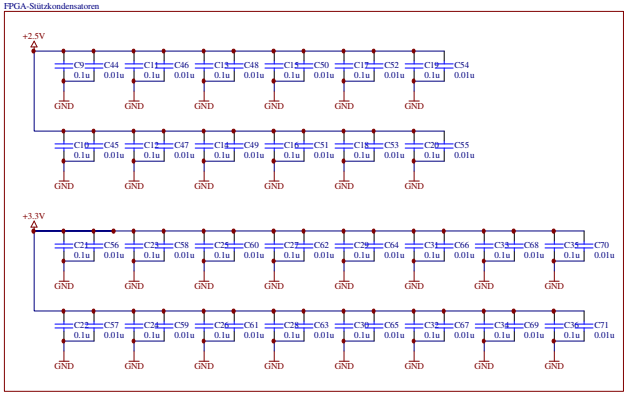




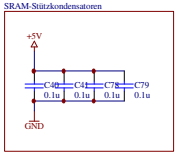




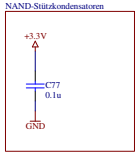
FPGA-Stützkondensatoren



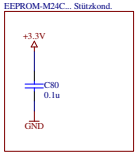
SRAM-Stützkondensatoren



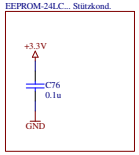
NAND-Stützkondensatoren



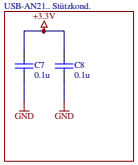
EEPROM-M24C... Stützkond.



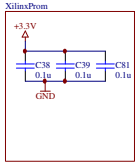
EEPROM-24LC... Stützkond.



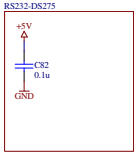
USB-AN21.. Stützkond.



XilinxProm



RS232-DS275



6. References

[JGBH01] Marcel Jacomet, Josef Goette, Jörg Breitenstein, Markus Hager, *On a Development Environment for Real-Time Information Processing in System-on-Chip Solutions*, IEEE 14th Symposium on Integrated Circuits and System Design, SBCCI'01, pp 28-31, Sept. 2001, Brasilia (also published in World Multiconference on Systemics, Cybernetics and Informatics, Volume XV: Industrial Systems, ISAS SCI'2001, July 22-25th, 2001, Orlando).

[JGHC02] Marcel Jacomet, Josef Goette, Markus Hager, William Chigutsa, Nicolas Leuba, *Hardware-in-the-Loop Simulation by linking Matlab/Simulink with a HW/SW Co-Design Rapid-Prototyping Platform*, proceedings of Designers Forum at IEEE Design Automation and Test in Europe, DATE'2002, Paris, March 3-8th, 2002.

[CCS] CCS Inc. C Compiler Manual for PIC processors, CCP Inc., http://www.ccsinfo.com

[Mic] Microchip Inc. Technical Data Sheets on PIC16C7X and PIC 16C5X series, http://www.microchip.com

[Mat] Mathworks Inc. Matlab/Simulink Tools, http://www.mathworks.com

[Syn] Synplicity Synthesis Tools, http://www.synplicity.com

[Xil] Xilinx Inc. System Generator Tools, http://www.xilinx.com